A course on MATLAB and It's Applications

Dr. M.Srinivasarao

9^{th} November 2009

Message from Vice Chancellor

I am extremely happy to say that the Anchor Institute – Chemicals & Petrochemicals Sector at this University started functioning from the day one, Government of Gujarat –Industries and Mines Department selected the University's faculty of Technology looking to its Chemical Engineering Department's strength.

I am very sure that the Anchor Institute will function in the most effective way to fulfill the expectations of Government of Gujarat – Industries & Mines Department(GOG-I&MD) in its proactive step to develop the manpower for the fastest growing Sector of Chemicals & Petrochemicals in the state and enhance the employability passing out students and capabilities of existing employed personnel in the industries & practicing Chemical Engineers.

I am pleased to learn that the Faculty members of not only Chemical Engineering Departments but of other Engineering Departments of Engineering Colleges have also responded very well to the announcement of the Anchor Institute's First training programme " MATLAb and its Applications". It also brought me happiness to learn that besides faculty members, post graduate students of Chemical Engineering Department have also shown interest in learning beyond their curriculum in this programme.

I wish the best for the programme and also appeal to all the participants to take utmost benefit in updating their knowledge, use the learning at your home Institutes in the benefit of your students and other faculty members. Please give your feedback at the end in the prescribed form to the Coordinator so that we will have more opportunities to improve and serve you better in future.

Dr. H. M. Desai,

Vice Chancellor,

Dharmsinh Desai University

Message from Dean, Faculty of Technology

Dr. H M Desai, The Vice Chancellor of Dharmsinh Desai Univeristy (DDU) and a leading Educationist took up the challenge of establishing a model Training Centre for Chemicals & Petrochemicals sector of Gujarat when the Industries Commisioner appealed to come forward to build and develop manpower for this fastest growing sector. Dr. Desai assigned the task to Faculty of Technology (FOT) and teammates of Chemical Engineering Department worked day and night, prepared the proposal in the shortest time and made presentations before the gethering of leading industrialists, educationist and government officials. Looking to the strengths of FOT, DDU and in particular its Chemical Engineering Department, the Industries & Mines Department selected DDU as Anchor Instittue for Chemicals & Petrochemicals sector. It was also decided that L.D. College of Engg. having Chemical Engg. Dept. will work as Co Anchor Institute for this sector. Both Anchor and CO-anchor Institutes will work under the guidance of Mentor Dr. Sanjay Mahajani, Professor of Chemical Engineering, IIT-B

The Department of Chem. Engg., DD University, is one of the oldest departments offering Chem. Engg. courses at Diploma, Degree, Postgraduate and Ph D .level. This unique feature has made it very versatile and competitive. Its alumni are very well placed not only in sectors of engineering but also in finance and management. A good number of them have become successful entrepreneur. The faculty has high degree of dedication and the work environment is conducive of enhancing learning potential of one and all.

With this, I let you all know that we, at Dharmsinh Desai Univeristy, Nadiad will put our whole hearted efforts to prove our worth in fulfilling the expectations of Government of Gujarat , Industries &Mines Department in its endeveour of building and developing the manpower for the Chemicals & Petrochemicals Sector. We will also welcome the other aspirants who wish to join hands with us to share the responsibility.

Dr. P.A. Joshi

Dean, Faculty of Technolgoy,

Preface

MATLAB is an integrated technical computing environment that combines numeric computation, advanced graphics and visualization, and a highlevel programming language. (– www.mathworks.com/products/matlab) That statement encapsulates the view of The MathWorks, Inc., the developer of MATLAB .

MATLAB contains hundreds of commands to do mathematics. You can use it to graph functions, solve equations, perform statistical tests, and do much more. It is a high-level programming language that can communicate with its cousins, e.g., FORTRAN and C. You can produce sound and animate graphics. You can do simulations and modeling (especially if you have access not just to basic MATLAB but also to its accessory SIMULINK). You can prepare materials for export to the World Wide Web. In addition, you can use MATLAB, in conjunction withth e word processing and desktop publishing features of Microsoft Word , to combine mathematical computations with text and graphics to produce a polished, integrated, and interactive document.

MATLAB is more than a fancy calculator; it is an extremely useful and versatile tool. Even if you only know a little about MATLAB, you can use it to accomplish wonderful things. The hard part, however, is figuring out which of the hundreds of commands, scores of help pages, and thousands of items of documentation you need to look at to start using it quickly and effectively.

The goal of this course material is to get you started using MATLAB successfully and quickly. We discuss the parts of MATLAB you need to know without overwhelming you with details. We help you avoid the rough spots. We give you examples of real uses of MATLAB that you can refer to when you're doing your own work. And we provide a handy reference to the most useful features of MATLAB. When you're finished with this course, you will be able to use MATLAB effectively. You'll also be ready to explore more of MATLAB on your own. You might not be a MATLAB expert when you finish this course, but you will be prepared to become one — if that's what you want.

In writing, we drew on our experience to provide important information as quickly as possible. The material contains a short, focused introduction to MATLAB. It contains

practice problems (withcomplete solutions) so you can test your knowledge. There are several illuminating sample examples that show you how MATLAB can be used in real-world applications.

Here is a detailed summary of the contents of the course material.

Chapter 1, Getting Started, describes how to start MATLAB on different platforms. It tells you how to enter commands, how to access online help, how to recognize the various MATLAB windows you will encounter, and how to exit the application.

Chapter 2, Data entry and analysis, shows you how to do elementary mathematics using MATLAB. This chapter also introduces some of the data analysis features which are essential for engineers.

Chapter 3, This chapter will introduce you to the basic window features of the application, to the small program files (M-files) that you will use to make most effective use of the software. .

Chapter 4, MATLAB Programming , introduces you to the programming features of MATLAB.

Chapter 5, SIMULINK and GUIs, consists of two parts. The first part describes the MATLAB companion software SIMULINK, a graphically oriented package for modeling, simulating, and analyzing dynamical systems. Many of the calculations that can be done with MATLAB can be done equally well with SIMULINK. Brief discussion on S-function is also presented. The second part contains an introduction to the construction and deployment of graphical user interfaces, that is, GUIs, using MATLAB.

Chapter 6, Introduces you the basics of system identification and parametric models. After this chapter you will be in a position to develop mathematical parametric models using MATLAB.

Chapter 7,Optimisation tool box of the Matlab is introduced to you in this chapter. This chapter also gives brief discussion about some basic optimisation algorthms.

Chapter 8, Troubleshooting, is the place to turn when anything goes wrong. Many common problems can be resolved by reading (and rereading) the advice in this chapter.

Next, we have Appendix that gives Practice Sets Though not a complete reference, this course material is a handy guide to some of the useful features of MATLAB.

Contents

Message from Vice Chancellor i

Message from Dean Faculty of Technology ii

Preface iii

List of Figures ix

List of Tables x

1 Introduction 1

1.1 Applications of Optimisation problems 1

1.2 Types of Optimization and Optimisation Problems 3

1.2.1 Static Optimization 3

1.2.2 Dynamic Optimization 4

2 Conventional optimisation techniques 7

2.1 Introduction 7

2.2 Search Methods 7

2.2.1 Method of Uniform search 7

2.2.2 Method of Uniform dichotomous search 8

2.2.3 Method of sequential dichotomos search 9

2.2.4 Fibonacci search Technique 9

2.3 UNCONSTRAINED OPTIMIZATION 11

2.3.1 Golden Search Method 11

2.3.2 Quadratic Approximation Method 12

2.3.3 Steepest Descent Method 14

2.3.4 Newton Method 16

2.4 CONSTRAINED OPTIMIZATION 16

2.4.1 Lagrange Multiplier Method 17

2.4.2 Penalty Function Method 17

2.5 MATLAB BUILT-IN ROUTINES FOR OPTIMIZATION 19

3 Linear Programming **21**
3.1 The Simplex Method 23
3.2 Infeasible Solution 28
3.3 Unbounded Solution 28
3.4 Multiple Solutions 31
3.4.1 Matlab code for Linear Programming (LP) 31

4 Nonlinear Programming **32**
4.1 Convex and Concave Functions 35

5 Discrete Optimization **38**
5.1 Tree and Network Representation 39
5.2 Branch-and-Bound for IP 42

6 Integrated Planning and Scheduling of processes **46**
6.1 Introduction 46
6.2 Plant optimization hierarchy 46
6.3 Planning 49
6.4 Scheduling 50
6.5 Plantwide Management and Optimization 55
6.6 Resent trends in scheduling 57
6.6.1 State-Task Network (STN): 59
6.6.2 Resource – Task Network (RTN): 61
6.6.3 Optimum batch schedules and problem formulations (MILP),MINLP B&B: 63
6.6.4 Multi-product batch plants: 65
6.6.5 Waste water minimization (Equalization tank super structure): . . 66
6.6.6 Selection of suitable equalization tanks for controlled flow: 68

7 Dynamic Programming **69**

8 Global Optimisation Techniques **72**
8.1 Introduction 72
8.2 Simulated Annealing 72
8.2.1 Introduction 72
8.3 GA 74

8.3.1 Introduction 74
8.3.2 Definition 75
8.3.3 Coding 75
8.3.4 Fitness 76
8.3.5 Operators in GA 76
8.4 Differential Evolution 78
8.4.1 Introduction 78
8.4.2 DE at a Glance 79
8.4.3 Applications of DE 82
8.5 Interval Mathematics 83
8.5.1 Introduction 83
8.5.2 Interval Analysis 83
8.5.3 Real examples 84
8.5.4 Interval numbers and arithmetic 87
8.5.5 Global optimization techniques 88
8.5.6 Constrained optimization 92
8.5.7 References 93

9 Appendix **95**
9.1 Practice session -1 95

List of Figures

2.1 Method of uniform search 8
2.2 9
2.3 10

3.1 Linear progamming Graphical representation 22
3.2 Feasible reagion and slack variables 25
3.3 Simplex tablau 25
3.4 Initial tableau for simplex example 26
3.5 Basic solution for simplex example 27
3.6 Infeasible LP 29
3.7 Initial tableau for infeasible problem 29
3.8 Second iteration tableau for infesible problem 29
3.9 Simplex tableau for unbounded solution 30
3.10 Simplex problem for unbounded solution 30

4.1 Nonlinear programming contour plot 33
4.2 Nonlinear program graphical representation 34
4.3 Nonlinear programming minimum 35
4.4 Nonlinear programming multiple minimum 36
4.5 Examples of convex and nonconvex sets 36

5.1 Isoperimetric problem discrete decisions 38
5.2 Feasible space for discrete isoperimetric problem 39
5.3 Cost of seperations 1000 Rs/year 40
5.4 Tree representation 41
5.5 Network representation 41
5.6 Tree representation and cost diagram 43
5.7 Deapth first enumeration 44
5.8 Breadth first enumeration 45

6.1 Gantt chart for the optimal multiproduct plant schedule 53

6.2 58
6.3 59
6.4 Recipe networks 60
6.5 STN representation 61
6.6 RTN representation 62
6.7 Super structure of equilisation tanks 68

8.1 Schematic diagram of DE 80

List of Tables

Chapter 1

Introduction

Optimization involves finding the minimum/maximum of an objective function f (x) subject to some constraint x ∈ S. If there is no constraint for x to satisfy or, equivalently, S is the universe—then it is called an unconstrained optimization; otherwise, it is a constrained optimization. In this chapter, we will cover several unconstrained optimization techniques such as the golden search method, the quadratic approximation method, the Nelder–Mead method, the steepest descent method, the Newton method, the simulated-annealing (SA) method, and the genetic algorithm (GA). As for constrained optimization, we will only introduce the MATLAB built-in routines together with the routines for unconstrained optimization. Note that we don't have to distinguish maximization and minimization because maximizing f (x) is equivalent to minimizing -f (x) and so, without loss of generality, we deal only with the minimization problems.

1.1 Applications of Optimisation problems

Optimization problems arise in almost all fields where numerical information is processed (science, engineering, mathematics, economics, commerce, etc.). In science, optimization problems arise in data fitting, variational principles, solution of differential and integral equations by expansion methods, etc. Engineering applications are in design problems, which usually have constraints in the sense

that variables cannot take arbitrary values. For example, while designing a bridge, an engineer will be interested in minimizing the cost, while maintaining a certain minimum strength for the structure. Optimizing the surface area for a given volume of a reactor is another example of constrained optimization. While most formulations of optimization problems require the global minimum to be found, mosi of the methods are only able to find a local minimum. A function has a local minimum, at a point where it assumes the lowest value in a small neighbourhood of the point, which is not at the boundary of that

neighbourhood.

To find a global minimum we normally try a heuristic approach where several local minima are found by repeated trials with different starting values or by using different techniques. The different starting values may be obtained by perturbing the local minimizers by appropriate amounts. The smallest of all known local minima is then assumed to be the global minimum. This procedure is obviously unreliable, since it is impossible to ensure that all local minima have been found. There is always the possibility that at some unknown local minimum, the function assumes an even smaller value. Further, there is no way of verifying that the point so obtained is indeed a global minimum, unless the value of the function at the global minimum is known independently. On the other hand, if a point i s claimed to be the solution of a system of non-linear equations, then it can, in principle, be verified by substituting in equations to check whether all the equations are satisfied or not. Of course, in practice, the round-off error introduces some uncertainty, but that can be overcome.

Owing to these reasons, minimization techniques are inherently unreliable and should be avoided if the problem can be reformulated to avoid optimization. However, there are problems for which no alternative solution method is known and we have to use these techniques. The following are some examples.

1. Not much can be said about the existence and uniqueness of either the

2. It is possible that no minimum of either type exists, when the function is

3. Even if the function is bounded from below, the minimum may not exist

4. Even if a minimum exists, it may not be unique; for exarnple,Xx) = sin x global or the local minimum of a function of several variables.

not bounded from below [e.g.,Ax) = XI. [e.g.,Ax) = e"]. has an infinite number of both local and global minima.

5. Further, infinite number of local minimum may exist, even when there is no global minimum [e.g.,Ax) = x + 2 sin x].

6. If the function or its derivative is not continuous, then the situation could be even more complicated. For example,Ax) = & has a global minimum at x = 0, which is not a local minimum [i.e.,Ax) = 01.

Optimization in chemical process industries infers the selection of equipment and operating conditions for the production of a given material so that the profit will be maximum. This could be interpreted as meaning the maximum output of a particular substance for a given capital outlay, or the minimum investment for a specified production rate. The former is a mathematical problem of evaluating the appropriate values of a set of variables to maximize a dependent variable, whereas the latter may be considered to be one of locating a minimum value. However, in terms of profit, both types of problems are

maximization problems, and the solution of both is generally accomplished by means of an economic balance (trade-@ between the capital and operating costs. Such a balance can be represented as shown in Fig.(??), in which the capital, the operating cost, and the total cost are plotted againstf, which is some function of the size of the equipment. It could be the actual size of the equipment; the number of pieces of equipment, such as the number of stirred tank reactors in a reactor battery; the frames in a filter press; some parameter related to the size of the equipment, such as the reflux ratio in a distillation unit; or the solvent-to-feed ratio in a solvent extraction process. Husain and Gangiah (1976) reported some of the optimization techniques that are used for chemical engineering applications.

1.2 Types of Optimization and Optimisation Problems

Optimization in the chemical field can be divided into two classes:

1. Static optimization
2. Dynamic optimization

1.2.1 Static Optimization

Static optimization is the establishment of the most suitable steady-state operation conditions of a process. These include the optimum size of equipment and production levels, in addition to temperatures, pressures, and flow rates. These can be established by setting up the best possible mathematical model of the process, which is maximized by some suitable technique to give the most favourable operating conditions. These conditions would be nominal conditions and would not take into account the fluctuations in the process about these nominal conditions.

With steady-state optimization (static Optimization), as its name implies, the process is assumed to be under steady-state conditions, and may instantaneously be moved to a new steady state, if changes in load conditions demand so, with the aid of a conventional or an optimization computer. Steady-state optimization is applicable to continuous processes, which attain a new steady state after a change in manipulated inputs within an acceptable time interval. The goal of static optimization is to develop and realize an optimum modelfor the process in question.

1.2.2 Dynamic Optimization

Dynamic optimization the establishment of the best procedure for correcting the fluctuations in a process used in the static optimization analysis. It requires knowledge of the dynamic characteristics of the equipment and also necessitates predicting the best way in which a change in the process conditions can be corrected. In reality, it is an extension of the automatic control analysis of a process.

As mentioned earlier, static optimization is applicable to continuous processes which attain a new steady state after a change in manipulated inputs within an acceptable time interval. With unsteady-state (dynamic) optimization, the objective is not only to maintain a process at an optimum level under steady-state conditions, but also to seek the best path for its transition from one steady state to another. The optimality function then becomes a time function, and the objective is to maximize or minimize the time-averaged performance criterion. Although similar to steadystate optimization in some respects, dynamic optimization is more elaborate because it involves a time-averaged function rather than individual quantities. The goal of control in this case is to select at any instant of time a set of manipulated variablesthat will cause the controlled system to behave in an optimum manner in the face of any set of disturbances.

Optimum behaviour is defined as a set of output variables ensuring the maximization of a certain objective or return function, or a change in the output variables over a definite time interval such that a predetermined functional value of these output variables is maximized or minimized.

As mentioned earlier, the goal of static optimization is to develop and realize an optimum model for the process in question, whereas dynamic optimization seeks to develop and realize an optimum control system for the process. In other words, static optimization is an optimum model for the process, whereas dynamic optimization is the optimum control system for the process.

Optimization is categorized into the following five groups:

1)Analytical methods

(a) Direct search (without constraints)

(b) Lagrangian multipliers (with constraints)

(c) Calculus of variations (examples include the solution of the Euler equation, optimum temperature conditions for reversible exothermic reactions in plug-flow beds, optimum temperature conditions for chemical reactors in the case of constraints on temperature range, Multilayer adiabatic reactors, etc.)

(d) Pontryagin's maximum principle (automatic control)

2. Mathematical programming:

(a) Geometric programming (algebraic functions)

(b) Linear programming (applications include the manufacture of products for maximum return from different raw materials, optimum

utilization of equipment, transportation problems, etc.)

(c) Dynamic programming (multistage processes such as distillation, extraction, absorption, cascade reactors, multistage adiabatic beds, interacting chain of reactors, etc.; Markov processes, etc.)

3. Gradient methods:

(a) Method of steepest descent (ascent)

(b) Sequential simplex method (applications include all forms of problems such as optimization of linear and non-linear functions

with and without linear and non-linear constraints, complex chemical engineering processes, single and cascaded interacting reactors)

4. Computer control and model adaptation:

5. Statistical optimization: All forms (complex chemical engineering systems)

(a) Regression analysis (non-deterministic systems)

(b) Correlation analysis (experimental optimization and designs: Brandon Though, for completeness, all the methods of optimization are listed above, let us restrict our discussion to some of the most important and widely used methods of optimization.

Optimization problems can be divided into the following broad categories depending on the type of decision variables, objective function(s), and constraints.

• Linear programming (LP): The objective function and constraints are linear. The decision variables involved are scalar and continuous.

• Nonlinear programming (NLP): The objective function and/or constraints are nonlinear. The decision variables are scalar and continuous.

• Integer programming (IP): The decision variables are scalars and integers.

• Mixed integer linear programming (MILP): The objective function and constraints are linear. The decision variables are scalar; some of them are integers whereas others are continuous variables.

• Mixed integer nonlinear programming (MINLP): A nonlinear programming problem involving integer as well as continuous decision variables.

• Discrete optimization: Problems involving discrete (integer) decision variables. This includes IP, MILP, and MINLPs.

• Optimal control: The decision variables are vectors.

• Stochastic programming or stochastic optimization: Also termed optimization under uncertainty. In these problems, the objective function and/or the constraints have uncertain (random) variables. Often involves the above categories as subcategories.

• Multiobjective optimization: Problems involving more than one objective. Often involves the above categories as subcategories.

Chapter 2

Conventional optimisation techniques

2.1 Introduction

In this chapter, we will introduce you some of the very well known conventional optimisation techniques. A very brief review of search methods followed by gradient based methods are presented. The compilation is no way exhaustive. Matlab programs for some of the optimisation techniques are also presented in this chapter.

2.2 Search Methods

Many a times the mathematical model for evaluation of the objective function is not available. To evaluate the value of the objective funciton an experimental run has to be conducted. Search procedures are used to determine the optimal value of the variable decision variable.

2.2.1 Method of Uniform search

Let us assume that we want to optimise the yield y and only four experiments are allowed due to certain plant conditions. An unimodel function can be represented as shown in the figure where the peak is at 4.5. This maximum is what we are going to find. The question is how close we can react to this optimum by systematic experimentation.

The most obvious way is to place the four experiments equidistance over the interval that is at 2,4,6 and 8. We can see from the figure that the value at 4 is higher than the value of y at 2. Since we are dealing with a unimodel function the optimal value can not lie between x=0 and x=2. By similar reasoning the area between x=8 and 10 can be

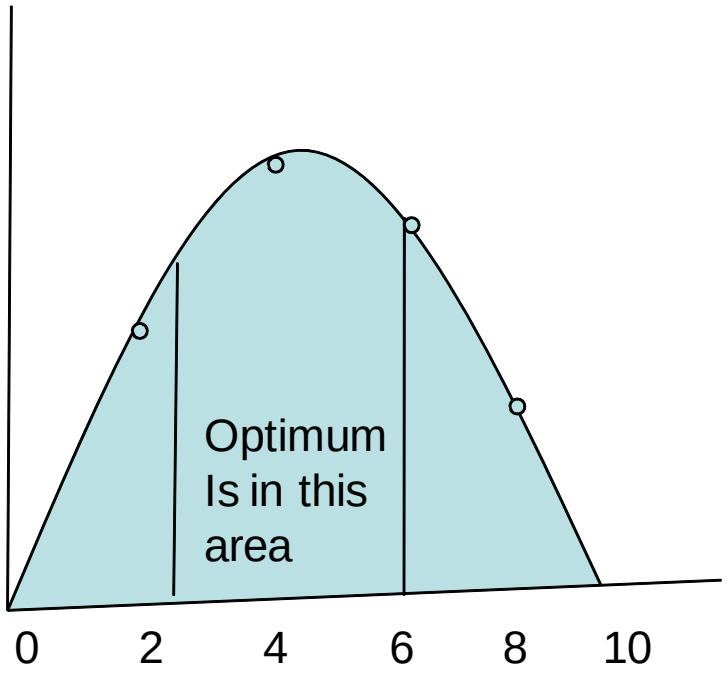


Figure 2.1: Method of uniform search

eleminated as well as that between 6 and 8. the area remaining is area between 2 and 6.

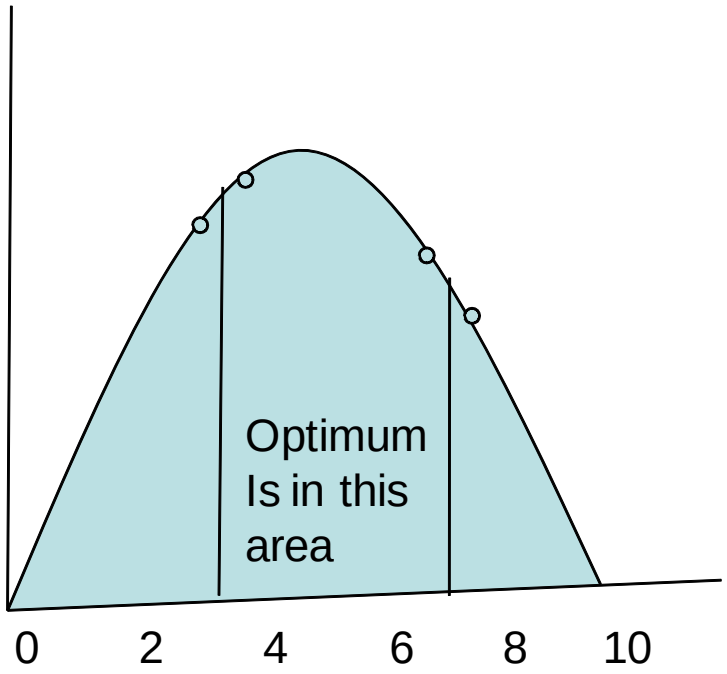
If we take the original interval as L and the F as fraction of original interval left after performing N experiments then N experiments devide the interval into N+1 intervals. Width of each interval is $\frac{L}{N+1}$ OPtimum can be specified in two of these intervals.That leaves 40% area in the given example

$$\begin{aligned} F &= \frac{2L}{N+1} * \frac{1}{L} = \frac{2}{N+1} \\ &= \frac{2}{4+1} = 0.4 \end{aligned}$$

2.2.2 Method of Uniform dichotomous search

The experiments are performed in pairs and these pairs are spaced evenly over the entire intervel. For the problem these pairs are specified as 3.33 and 6.66. From the figure it can be seen that function around 3.33 it can be observed that the optimum does not lie between 0 and 3.33 and similarly it does not lie between 6.66 and 10. The total area left is between 3.33 and 6.66. The original region is devided into $\frac{N}{2}+1$ intervals of the width $\frac{L}{\frac{N}{2}+1}$ The optimum is location in the width of one interval There fore

$$\begin{aligned} F &= \frac{L}{\frac{N}{2}+1} * \frac{1}{L} = \frac{2}{N+2} \\ &= \frac{2}{4+2} = 0.33 \end{aligned}$$



2.2.3 Method of sequential dichotomos search

The sequential search is one where the investigator uses the information availble from the previous experiments before performing the next experiment. Inour exaple perform the search around the middle of the search space. From the infromation available discard the region between 5 and 10. Then perform experiment between 2.5 and discard the region between 0 and 2.5 the region lect out is between 2.5 and 5 only. This way the fraction left out after each set of experiments become half that of the region left out. it implies that

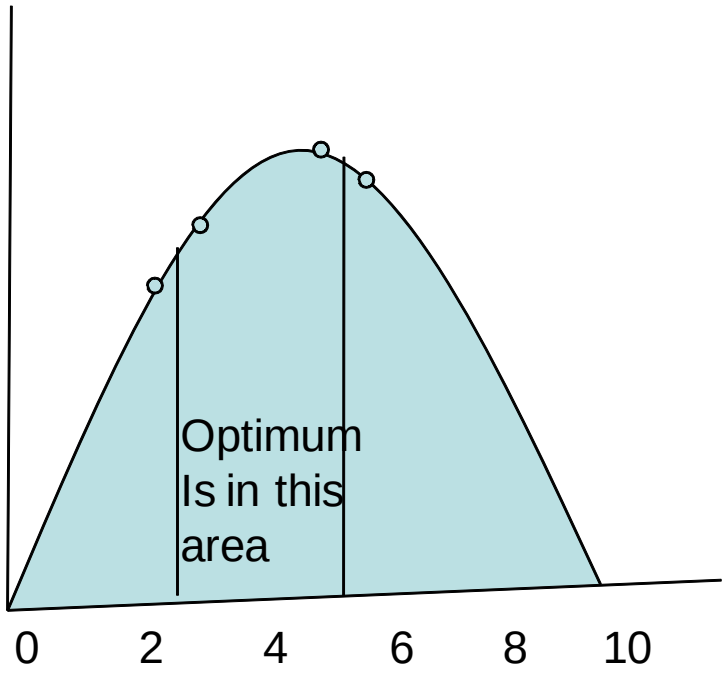
$$\begin{aligned} F &= \frac{1}{2^{\frac{N}{2}}} \\ &= \frac{1}{2^2} = 0.25 \end{aligned}$$

2.2.4 Fibonacci search Technique

A more efficient sequential search technique is Fibonacci techniquie. The Fibonacci series is considered as Note that the nuber in the sequence is sum of previous two numbers.

$$\begin{aligned} xn &= x_{n-1} + x_{n-2} \\ n &\succeq 2 \end{aligned}$$

The series is 1 2 3 5 8 10 ... To perform the search a pair of experiments are performed



equidistance from each end of the interval. The distance d1 is determined from the follwing expression.

$$d_1 = \frac{F_{N-2}}{F_{N-1}} L$$

Where N is number of experiments and L is total length. IN our problem L is 10, N is 4 and F_{N-2} is 2 and F_N=5 . First two experiments are run 4 units away from each end. From the result the area between 6 and 10 can be eliminated.

$$d_1 = \frac{2}{5} 10 = 4$$

The area remaining is between 0 and 6 the new length will be 6 and new value of d2 is obtained by substituting N-1 for N

$$d_2 = \frac{F_{N-3}}{F_{N-1}} L = \frac{F_1}{F_3} L = \frac{1}{3} 6 = 2$$

The next pair of experiments are performed around 2 and the experiment at 4 need not be performed as we have allready done it. This is the advantage of Fibonacci search the remaining experiment can be performed as dicocomos search to identify the optimal reagion around 4. This terns out to be the region between 4 and 6. The fraction left out is

$$F = \frac{1}{F_N} = \frac{1}{F_4} = \frac{1}{5} = 0.2$$

2.3 UNCONSTRAINED OPTIMIZATION

2.3.1 Golden Search Method

This method is applicable to an unconstrained minimization problem such that the solution interval [a, b] is known and the objective function f (x) is unimodal within the interval; that is, the sign of its derivative f (x) changes at most once in [a, b] so that f (x) decreases/increases monotonically for [a, xo]/[xo, b], where xo is the solution that we are looking for. The so-called golden search procedure is summarized below and is cast into the routine "opt_gs()".We made a MATLAB program "nm711.m", which uses this routine to find the minimum point of the

objective function

$$f(x) = \frac{(x2-4)^2}{8} - 1 \tag{2.1}$$

GOLDEN SEARCH PROCEDURE

Step 1. Pick up the two points c = a + (1 - r)h and d = a + rh inside the interval [a, b], where r = ($\sqrt{5}$ - 1)/2 and h = b - a.

Step 2. If the values of f (x) at the two points are almost equal [i.e., f (a) $\approx$ f (b)] and the width of the interval is sufficiently small (i.e., h $\approx$ 0), then stop the iteration to exit the loop and declare xo = c or xo = d depending on whether f (c) < f(d) or not. Otherwise, go to Step 3.

Step 3. If f (c) < f(d), let the new upper bound of the interval b to d; otherwise, let the new lower bound of the interval a to c. Then, go to Step 1.

```
function [xo,fo] = opt_gs(f,a,b,r,TolX,TolFun,k)
h = b - a; rh = r*h; c = b - rh; d = a + rh;
fc = feval(f,c); fd = feval(f,d);
if k <= 0 | (abs(h) < TolX & abs(fc - fd) < TolFun)
if fc <= fd, xo = c; fo = fc;
else xo = d; fo = fd;
end
if k == 0, fprintf('Just the best in given # of iterations'), end
else
if fc < fd, [xo,fo] = opt_gs(f,a,d,r,TolX,TolFun,k - 1);
else [xo,fo] = opt_gs(f,c,b,r,TolX,TolFun,k - 1);
end
end
%nm711.m to perform the golden search method
```

f711 = inline('(x.*x-4).^2/8-1','x');

a = 0; b = 3; r =(sqrt(5)-1)/2; TolX = 1e-4; TolFun = 1e-4; MaxIter = 100;

[xo,fo] = opt_gs(f711,a,b,r,TolX,TolFun,MaxIter)

Note the following points about the golden search procedure.

At every iteration, the new interval width is

$$b - c = b - (a + (1 - r)(b - a)) = rhord - a = a + rh - a = rh$$

so that it becomes r times the old interval width (b - a = h).

The golden ratio r is fixed so that a point c1 = b1 - rh1 = b - r2h in the new interval [c, b] conforms with d = a + rh = b - (1 - r)h, that is,

$$r^2 = 1 - r,\ r^2 + r - 1 = 0, \tag{2.2}$$

$$r = \frac{-1 + \sqrt{1 + 4}}{2} = \frac{-1 + \sqrt{5}}{2} \tag{2.3}$$

2.3.2 Quadratic Approximation Method

The idea of this method is to (a) approximate the objective function f (x) by a quadratic function p2(x) matching the previous three (estimated solution) points and (b) keep updating the three points by replacing one of them with the minimum point of p2(x). More specifically, for the three points

{(x0, f0), (x1, f1), (x2, f2)} with x0 < x1 < x2

we find the interpolation polynomial p2(x) of degree 2 to fit them and replace one of them with the zero of the derivative—that is, the root of p'2 (x) = 0 :

$$x = x3 = \frac{f_o\,(x_1^2 - x_2^2) + f_1\,(x_2^2 - x_0^2) + f_2\,(x_0^2 - x_1^2)}{2\,[_o\,(x_1 - x_2) + f_1\,(x_2 - x_0) + f_2\,(x_0 - x_1)]} \tag{2.4}$$

In particular, if the previous estimated solution points are equidistant with an equal distance h (i.e., x2 - x1 = x1 - x0 = h), then this formula becomes

$$x = x3 = \frac{f_o\,(x_1^2 - x_2^2) + f_1\,(x_2^2 - x_0^2) + f_2\,(x_0^2 - x_1^2)}{2\,[_o\,(x_1 - x_2) + f_1\,(x_2 - x_0) + f_2\,(x_0 - x_1)]}\Big|_{\substack{x1=x+h \\ x2=x1+h}} \tag{2.5}$$

We keep updating the three points this way until |x2 - x0| ≈ 0 and/or |f (x2) - f (x0)| ≈ 0, when we stop the iteration and declare x3 as the minimum point. The rule for updating the three points is as follows.

1. In case x0 < x3 < x1, we take {x0, x3, x1} or {x3, x1, x2} as the new set of three points depending on whether f (x3) < f(x1) or not.

2. In case x1 < x3 < x2, we take {x1, x3, x2} or {x0, x1, x3} as the new set of three points depending on whether f (x3) ≤ f (x1) or not.

This procedure, called the quadratic approximation method, is cast into the MATLAB routine "opt_quad()", which has the nested (recursive call) structure. We made the MATLAB program "nm712.m", which uses this routine to find the minimum point of the objective function (7.1.1) and also uses the MATLAB built-in routine "fminbnd()" to find it for cross-check.

(cf) The MATLAB built-in routine "fminbnd()" corresponds to "fmin()" in the MATLAB of version.5.x.

```
function [xo,fo] = opt_quad(f,x0,TolX,TolFun,MaxIter)
%search for the minimum of f(x) by quadratic approximation method
if length(x0) > 2, x012 = x0(1:3);
else
if length(x0) == 2, a = x0(1); b = x0(2);
else a = x0 - 10; b = x0 + 10;
end
x012 = [a (a + b)/2 b];
end
f012 = f(x012);
[xo,fo] = opt_quad0(f,x012,f012,TolX,TolFun,MaxIter);
function [xo,fo] = opt_quad0(f,x012,f012,TolX,TolFun,k)
x0 = x012(1); x1 = x012(2); x2 = x012(3);
f0 = f012(1); f1 = f012(2); f2 = f012(3);
nd = [f0 - f2 f1 - f0 f2 - f1]*[x1*x1 x2*x2 x0*x0; x1 x2 x0]';
x3 = nd(1)/2/nd(2); f3 = feval(f,x3); %Eq.(7.1.4)
if k <= 0 | abs(x3 - x1) < TolX | abs(f3 - f1) < TolFun
xo = x3; fo = f3;
if k == 0, fprintf('Just the best in given # of iterations'), end
else
if x3 < x1
if f3 < f1, x012 = [x0 x3 x1]; f012 = [f0 f3 f1];
else x012 = [x3 x1 x2]; f012 = [f3 f1 f2];
end
else
if f3 <= f1, x012 = [x1 x3 x2]; f012 = [f1 f3 f2];
else x012 = [x0 x1 x3]; f012 = [f0 f1 f3];
end
```

end
[xo,fo] = opt_quad0(f,x012,f012,TolX,TolFun,k - 1);
end
%nm712.m to perform the quadratic approximation method
clear, clf
f711 = inline('(x.*x - 4).^2/8-1', 'x');
a = 0; b = 3; TolX = 1e-5; TolFun = 1e-8; MaxIter = 100;
[xoq,foq] = opt_quad(f711,[a b],TolX,TolFun,MaxIter)
%minimum point and its function value
[xob,fob] = fminbnd(f711,a,b) %MATLAB built-in function

2.3.3 Steepest Descent Method

This method searches for the minimum of an N-dimensional objective function in the direction of a negative gradient

$$-g(x) = -\nabla f(x) = -\left[\frac{\partial f(x)}{\partial x_1}\frac{\partial f(x)}{\partial x_2}....\frac{\partial f(x)}{\partial x_N}\right]^T \tag{2.6}$$

with the step-size α_k (at iteration k) adjusted so that the function value is minimized along the direction by a (one-dimensional) line search technique like the quadratic approximation method. The algorithm of the steepest descent method is summarized in the following box and cast into the MATLAB routine "opt_steep()".

We made the MATLAB program "nm714.m" to minimize the objective function (7.1.6) by using the steepest descent method.

STEEPEST DESCENT ALGORITHM

Step 0. With the iteration number k = 0, find the function value f0 = f (x0) for the initial point x0.

Step 1. Increment the iteration number k by one, find the step-size α_{k-1} along the direction of the negative gradient -gk-1 by a (one-dimensional) line search like the quadratic approximation method.

$$\blacksquare_{k-1} = ArgMin_\alpha f(x_{k-1} - \frac{\alpha g_{k-1}}{||gk-1||}) \tag{2.7}$$

Step 2. Move the approximate minimum by the step-size αk-1 along the direction of the negative gradient -gk-1 to get the next point

$$xk = xk - 1 - \alpha k - 1gk - 1/||gk - 1||$$

Step 3. If xk ≈ xk-1 and f (xk)≈ f (xk-1), then declare xk to be the minimum and terminate the procedure. Otherwise, go back to step 1.

```
function [xo,fo] = opt_steep(f,x0,TolX,TolFun,alpha0,MaxIter)
% minimize the ftn f by the steepest descent method.
%input: f = ftn to be given as a string 'f'
% x0 = the initial guess of the solution
%output: x0 = the minimum point reached
% f0 = f(x(0))
if nargin < 6, MaxIter = 100; end %maximum # of iteration
if nargin < 5, alpha0 = 10; end %initial step size
if nargin < 4, TolFun = 1e-8; end %|f(x)| < TolFun wanted
if nargin < 3, TolX = 1e-6; end %|x(k)- x(k - 1)|<TolX wanted
x = x0; fx0 = feval(f,x0); fx = fx0;
alpha = alpha0; kmax1 = 25;
warning = 0; %the # of vain wanderings to find the optimum step size
for k = 1: MaxIter
g = grad(f,x); g = g/norm(g); %gradient as a row vector
alpha = alpha*2; %for trial move in negative gradient direction
fx1 = feval(f,x - alpha*2*g);
for k1 = 1:kmax1 %find the optimum step size(alpha) by line search
fx2 = fx1; fx1 = feval(f,x-alpha*g);
if fx0 > fx1+TolFun & fx1 < fx2 - TolFun %fx0 > fx1 < fx2
den = 4*fx1 - 2*fx0 - 2*fx2; num = den - fx0 + fx2; %Eq.(7.1.5)
alpha = alpha*num/den;
x = x - alpha*g; fx = feval(f,x); %Eq.(7.1.9)
break;
else alpha = alpha/2;
end
end
if k1 >= kmax1, warning = warning + 1; %failed to find optimum step
size
else warning = 0;
end
if warning >= 2|(norm(x - x0) < TolX&abs(fx - fx0) < TolFun), break; end
x0 = x; fx0 = fx;
end
xo = x; fo = fx;
if k == MaxIter, fprintf('Just best in %d iterations',MaxIter), end
%nm714
```

f713 = inline('x(1)*(x(1) - 4 - x(2)) + x(2)*(x(2)- 1)','x');
x0 = [0 0], TolX = 1e-4; TolFun = 1e-9; alpha0 = 1; MaxIter = 100;
[xo,fo] = opt_steep(f713,x0,TolX,TolFun,alpha0,MaxIter)

2.3.4 Newton Method

Like the steepest descent method, this method also uses the gradient to search for the minimum point of an objective function. Such gradient-based optimization methods are supposed to reach a point at which the gradient is (close to) zero. In this context, the optimization of an objective function f (x) is equivalent to finding a zero of its gradient g(x), which in general is a vector-valued function of a vector-valued independent variable x. Therefore, if we have the gradient function g(x) of the objective function f (x), we can solve the system of nonlinear equations g(x) = 0 to get the minimum of f (x) by using the Newton method.

The matlabcode for the same is provided below

xo = [3.0000 2.0000], ans = -7
%nm715 to minimize an objective ftn f(x) by the Newton method.
clear, clf
f713 = inline('x(1).^2 - 4*x(1) - x(1).*x(2) + x(2).^2 - x(2)','x');
g713 = inline('[2*x(1) - x(2) - 4 2*x(2) - x(1) - 1]','x');
x0 = [0 0], TolX = 1e-4; TolFun = 1e-6; MaxIter = 50;
[xo,go,xx] = newtons(g713,x0,TolX,MaxIter);
xo, f713(xo) %an extremum point reached and its function value

The Newton method is usually more efficient than the steepest descent method if only it works as illustrated above, but it is not guaranteed to reach the minimum point. The decisive weak point of the Newton method is that it may approach one of the extrema having zero gradient, which is not necessarily a (local) minimum, but possibly a maximum or a saddle point.

2.4 CONSTRAINED OPTIMIZATION

In this section, only the concept of constrained optimization is introduced.

2.4.1 Lagrange Multiplier Method

A class of common optimization problems subject to equality constraints may be nicely handled by the Lagrange multiplier method. Consider an optimization problem with M equality constraints.

$$Minf(x) \tag{2.8}$$

$$h(x) = \begin{matrix} h_1(x) \\ h_2(x) \\ h_3(x) \\ . \\ h_4(x) \end{matrix} = 0$$

According to the Lagrange multiplier method, this problem can be converted to the following unconstrained optimization problem:

$$Minl(x, \lambda) = f(x) + \lambda^T h(x) = f(x) + \sum_{m=1}^{M} \lambda_m h_{|m}(x)$$

The solution of this problem, if it exists, can be obtained by setting the derivatives of this new objective function l(x, λ) with respect to x and λ to zero: Note that the solutions for this system of equations are the extrema of the objective function. We may know if they are minima/maxima, from the positive/negative- definiteness of the second derivative (Hessian matrix) of l(x, λ) with respect to x.

Inequality Constraints with the Lagrange Multiplier Method. Even though the optimization problem involves inequality constraints like gj (x) $\leq$ 0, we can convert them to equality constraints by introducing the (nonnegative) slack variables y_j^2

$$g_j(x) + y_j^2 = 0 \tag{2.9}$$

Then, we can use the Lagrange multiplier method to handle it like an equalityconstrained problem.

2.4.2 Penalty Function Method

This method is practically very useful for dealing with the general constrained optimization problems involving equality/inequality constraints. It is really attractive for optimization problems with fuzzy or loose constraints that are not so strict with zero tolerance.

The penalty function method consists of two steps. The first step is to construct a new objective function by including the constraint terms in such a way that violating the constraints would be penalized through the large value of the constraint terms in the objective function, while satisfying the constraints would not affect the objective function.

The second step is to minimize the new objective function with no constraints by using the method that is applicable to unconstrained optimization problems, but a non-gradient-based approach like the Nelder method. Why don't we use a gradient-based optimization method? Because the inequality constraint terms vmΨm(gm(x)) attached to the objective function are often determined to be zero as long as x stays inside the (permissible) region satisfying the corresponding constraint (gm(x) $\leq$ 0) and to increase very steeply (like Ψm(gm(x)) = exp(emgm(x)) as x goes out of the region; consequently, the gradient of the new objective function may not carry useful information about the direction along which the value of the objective function decreases.

From an application point of view, it might be a good feature of this method that we can make the weighting coefficient (wm,vm, and em) on each penalizing constraint term either large or small depending on how strictly it should be satisfied.

The Matlab code for this method is given below

```
%nm722 for Ex.7.3
% to solve a constrained optimization problem by penalty ftn method.
clear, clf
f ='f722p';
x0=[0.4 0.5]
TolX = 1e-4; TolFun = 1e-9; alpha0 = 1;
[xo_Nelder,fo_Nelder] = opt_Nelder(f,x0) %Nelder method
[fc_Nelder,fo_Nelder,co_Nelder] = f722p(xo_Nelder) %its results
[xo_s,fo_s] = fminsearch(f,x0) %MATLAB built-in fminsearch()
[fc_s,fo_s,co_s] = f722p(xo_s) %its results
% including how the constraints are satisfied or violated
xo_steep = opt_steep(f,x0,TolX,TolFun,alpha0) %steepest descent method
[fc_steep,fo_steep,co_steep] = f722p(xo_steep) %its results
[xo_u,fo_u] = fminunc(f,x0); % MATLAB built-in fminunc()
[fc_u,fo_u,co_u] = f722p(xo_u) %its results
function [fc,f,c] = f722p(x)
f=((x(1)+ 1.5)^2 + 5*(x(2)- 1.7)^2)*((x(1)- 1.4)^2 + .6*(x(2)-.5)^2);
c=[-x(1); -x(2); 3*x(1) - x(1)*x(2) + 4*x(2) - 7;
2*x(1)+ x(2) - 3; 3*x(1) - 4*x(2)^2 - 4*x(2)]; %constraint vector
v=[1 1 1 1 1]; e = [1 1 1 1 1]'; %weighting coefficient vector
fc = f +v*((c > 0).*exp(e.*c)); %new objective function
```

2.5 MATLAB BUILT-IN ROUTINES FOR OPTI-

MIZATION

In this section, we introduce some MATLAB built-in unconstrained optimization routinesincluding "fminsearch()" and "fminunc()" to the same problem, expecting that their nuances will be clarified. Our intention is not to compare or evaluate the performances of these sophisticated routines, but rather to give the readers some feelings for their functional differences.We also introduce the routine "linprog()"implementing Linear Programming (LP) scheme and "fmincon()" designed for attacking the (most challenging) constrained optimization problems. Interested readers are encouraged to run the tutorial routines "optdemo" or "tutdemo", which demonstrate the usages and performances of the representative built-in optimization routines such as "fminunc()" and "fmincon()".

```
%nm731_1
% to minimize an objective function f(x) by various methods.
clear, clf
% An objective function and its gradient function
f = inline('(x(1) - 0.5).^2.*(x(1) + 1).^2 + (x(2)+1).^2.*(x(2) - 1).^2','x');
g0 = '[2*(x(1)- 0.5)*(x(1)+ 1)*(2*x(1)+ 0.5) 4*(x(2)^2 - 1).*x(2)]';
g = inline(g0,'x');
x0 = [0 0.5] %initial guess
[xon,fon] = opt_Nelder(f,x0) %min point, its ftn value by opt_Nelder
[xos,fos] = fminsearch(f,x0) %min point, its ftn value by fminsearch()
[xost,fost] = opt_steep(f,x0) %min point, its ftn value by opt_steep()
TolX = 1e-4; MaxIter = 100;
xont = Newtons(g,x0,TolX,MaxIter);
xont,f(xont) %minimum point and its function value by Newtons()
[xocg,focg] = opt_conjg(f,x0) %min point, its ftn value by opt_conjg()
[xou,fou] = fminunc(f,x0) %min point, its ftn value by fminunc()
```

For constraint optimisation

```
%nm732_1 to solve a constrained optimization problem by fmincon()
clear, clf
ftn='((x(1) + 1.5)^2 + 5*(x(2) - 1.7)^2)*((x(1)-1.4)^2 + .6*(x(2)-.5)^2)';
f722o = inline(ftn,'x');
x0 = [0 0.5] %initial guess
A = []; B = []; Aeq = []; Beq = []; %no linear constraints
l = -inf*ones(size(x0)); u = inf*ones(size(x0)); % no lower/upperbound
options = optimset('LargeScale','off'); %just [] is OK.
[xo_con,fo_con] = fmincon(f722o,x0,A,B,Aeq,Beq,l,u,'f722c',options)
```

[co,ceqo] = f722c(xo_con) % to see how constraints are.

Chapter 3

Linear Programming

Linear programming (LP) problems involve linear objective function and linear constraints, as shown below in Example below.

Example: Solvents are extensively used as process materials (e.g., extractive agents) or process fluids (e.g., CFC) in chemical process industries. Cost is a main consideration in selecting solvents. A chemical manufacturer is accustomed to a raw material X1 as the solvent in his plant. Suddenly, he found out that he can effectively use a blend of X1 and X2 for the same purpose. X1 can be purchased at $4 per ton, however, X2 is an environmentally toxic material which can be obtained from other manufacturers. With the current environmental policy, this results in a credit of $1 per ton of X2 consumed.

He buys the material a day in advance and stores it. The daily availability of these two materials is restricted by two constraints: (1) the combined storage (intermediate) capacity for X1 and X2 is 8 tons per day. The daily availability for X1 is twice the required amount. X2 is generally purchased as needed. (2) The maximum availability of X2 is 5 tons per day. Safety conditions demand that the amount of X1 cannot exceed the amount of X2 by more than 4 tons. The manufacturer wants to determine the amount of each raw material required to reduce the cost of solvents to a minimum. Formulate the problem as an optimization problem. Solution: Let x1 be the amount of X1 and x2 be the amount of X2 required per day in the plant. Then, the problem can be formulated as a linear programming problem as given below.

$$\min_{x_1,x_2} 4x1 - x2 \tag{3.1}$$

Subject to

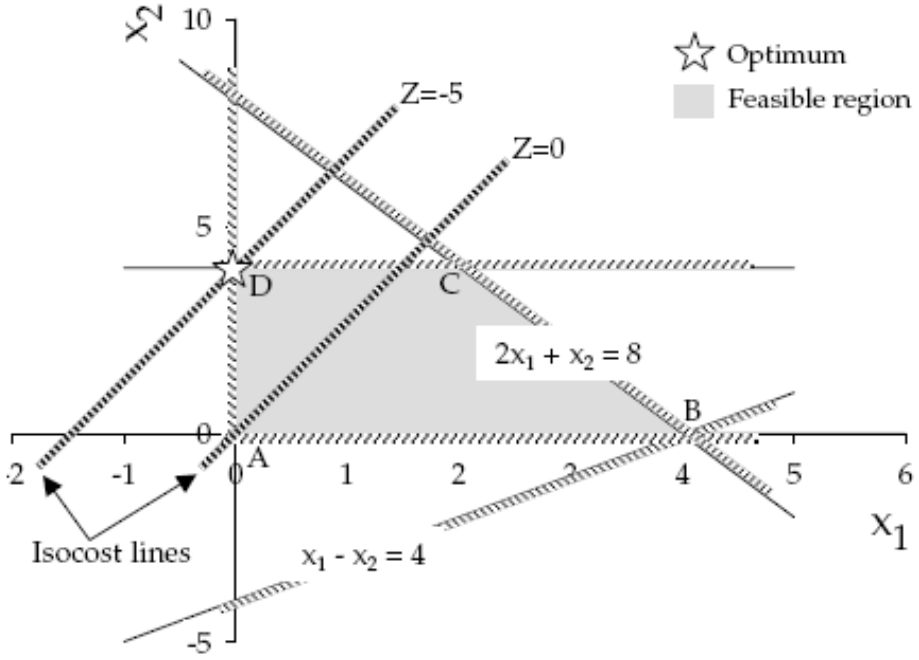


Figure 3.1: Linear progamming Graphical representation

$$2x1 + x2 \leq 8 \text{ storage constraint} \tag{3.2}$$

$$x2 \leq 5\text{Availability constraint} \tag{3.3}$$

$$x1 - x2 \leq 4 \text{ Safety constraint} \tag{3.4}$$

$$x1 \succeq 0 \tag{3.5}$$

$$x2 \succeq 0 \tag{3.6}$$

As shown above, the problem is a two-variable LP problem, which can be easily represented in a graphical form. Figure 2.1 shows constraints (2.2) through (2.4), plotted as three lines by considering the three constraints as equality constraints. Therefore, these lines represent the boundaries of the inequality constraints. In the figure, the inequality is represented by the points on the other side of the hatched lines. The objective function lines are represented as dashed lines (isocost lines). It can be seen that the optimal solution is at the point x1 = 0; x2 = 5, a point at the intersection of constraint (2.3) and one of the isocost lines. All isocost lines intersect constraints either once or twice. The LP optimum lies at a vertex of the feasible region, which forms the basis of the simplex method. The simplex method is a numerical optimization method for solving linear programming problems developed by George Dantzig in 1947.

3.1 The Simplex Method

The graphical method shown above can be used for two-dimensional problems; however, real-life LPs consist of many variables, and to solve these linear programming problems, one has to resort to a numerical optimization method such as the simplex method. The generalized form of an LP can be written as follows.

$$Optimize\ \mathrm{Z} = \sum_{i=1}^{n} \mathrm{Ci}\, xi \tag{3.7}$$

Subject to

$$\sum_{i=1}^{n} \mathrm{Ci}\, xi \leq bj \tag{3.8}$$

$$j = 1, 2, 3, ..., m \tag{3.9}$$

$$xj \in R \tag{3.10}$$

a numerical optimization method involves an iterative procedure. The simplex method involves moving from one extreme point

on the boundary (vertex) of the feasible region to another along the edges of the boundary iteratively. This involves identifying the constraints (lines) on which the solution will lie. In simplex, a slack variable is incorporated in every constraint to make the constraint an equality. Now, the aim is to solve the linear equations (equalities) for the decision variables x, and the slack variables s. The active constraints are then identified based on the fact that, for these constraints, the corresponding slack variables are zero.

The simplex method is based on the Gauss elimination procedure of solving linear equations. However, some complicating factors enter in this procedure: (1) all variables are required to be nonnegative because this ensures that the feasible solution can be obtained easily by a simple ratio test (Step 4 of the iterative procedure described below); and (2) we are optimizing the linear objective function, so at each step we want ensure that there is an improvement in the value of the objective function (Step 3 of the iterative procedure given below).

The simplex method uses the following steps iteratively.

Convert the LP into the standard LP form.

Standard LP

All the constraints are equations with a nonnegative right-hand side.

All variables are nonnegative.

– Convert all negative variables x to nonnegative variables using two variables (e.g., x = x+-x-); this is equivalent to saying if x = -5

then -5 = 5 - 10, x+ = 5, and x- = 10.

– Convert all inequalities into equalities by adding slack variables (nonnegative) for less than or equal to constraints (■) and by

subtracting surplus variables for greater than or equal to constraints (■).

The objective function must be minimization or maximization.

The standard LP involving m equations and n unknowns has m basic variables and n-m nonbasic or zero variables. This is explained below using Example

Consider Example in the standard LP form with slack variables, as given below.

Standard LP:

$$MTinimize - \ \mathrm{Z} \tag{3.11}$$

Subject to

$$\begin{aligned} -Z + 4x1 - x2 &= 0 && (3.12)\\ 2x1 + x2 + s1 &= 8 && (3.13)\\ x2 + s2 &= 5 && (3.14)\\ x1 - x2 + s3 &= 4 && (3.15)\\ x1 &\succeq 0; x2 \succeq 0 && (3.16)\\ s1 &\succeq 0; s2 \succeq 0; s3 \succeq 0 && (3.17) \end{aligned}$$

The feasible region for this problem is represented by the region ABCD in Figure . Table shows all the vertices of this region and the corresponding slack variables calculated using the constraints given by Equations (note that the nonnegativity constraint on the variables is not included).

It can be seen from Table that at each extreme point of the feasible region, there are n - m = 2 variables that are zero and m = 3 variables that are nonnegative. An extreme point of the linear program is characterized by these m basic variables.

In simplex the feasible region shown in Table gets transformed into a tableau

Determine the starting feasible solution. A basic solution is obtained by setting n - m variables equal to zero and solving for the values of theremaining m variables.

3. Select an entering variable (in the list of nonbasic variables) using the optimality (defined as better than the current solution) condition; that is, choose the next operation so that it will improve the objective function.

Stop if there is no entering variable.

Optimality Condition:

Point	x_1	x_2	s_1	s_2	s_3
A	0.0	0.0	8.0	5.0	4.0
B	4.0	0.0	0.0	5.0	0.0
C	1.5	5.0	0.0	0.0	7.5
D	0.0	5.0	3.0	0.0	9.0

Figure 3.2: Feasible reagion and slack variables

Row	$-Z$	x_1	x_2	s_1	s_2	s_3	RHS	Basic
0	1	4	-1	0	0	0	0	$-Z = 0$
1	0	2	1	1	0	0	8	$s_1 = 8$
2	0	0	$\underline{1}$	0	1	0	5	$s_2 = 5$
3	0	1	-1	0	0	1	4	$s_3 = 4$

Figure 3.3: Simplex tablau

• Entering variable: The nonbasic variable that would increase the objective function (for maximization). This corresponds to the nonbasic variable having the most negative coefficient in the objective function equation or the row zero of the simplex tableau. In many implementations of simplex, instead of wasting the computation time in finding the most negative coefficient, any negative coefficientin the objective function equation is used.

4. Select a leaving variable using the feasibility condition.

Feasibility Condition:

• Leaving variable: The basic variable that is leaving the list of basic variables and becoming nonbasic. The variable corresponding to the smallest nonnegative ratio (the right-hand side of the constraint divided by the constraint coefficient of the entering variable).

5. Determine the new basic solution by using the appropriate Gauss–Jordan

Row Operation.

Gauss–Jordon Row Operation:

• Pivot Column: Associated with the row operation.

• Pivot Row: Associated with the leaving variable.

• Pivot Element: Intersection of Pivot row and Pivot Column.

ROW OPERATION

• Pivot Row = Current Pivot Row $\div$ $PivotElement$.

Row	$-Z$	x_1	x_2	s_1	s_2	s_3	RHS	Basic	Ratio
0	1	4	-1	0	0	0	0	$-Z = 0$	—
1	0	2	1	1	0	0	8	$s_1 = 8$	8
2	0	0	$\underline{1}$	0	1	0	5	$s_2 = 5$	$\underline{5}$
3	0	1	-1	0	0	1	4	$s_3 = 4$	—

Figure 3.4: Initial tableau for simplex example

• All other rows: New Row = Current Row - (its Pivot Column Coefficients x New Pivot Row).

6. Go to Step 2.

To solve the problem discussed above using simplex method Convert the LP into the standard LP form. For simplicity, we are converting this minimization problem to a maximization problem with -Z as the objective function. Furthermore, nonnegative slack variables s1, s2, and s3 are added to each constraint.

$$MTinimize - \text{ Z} \tag{3.18}$$

Subject to

$$-Z + 4x1 - x2 = 0 \tag{3.19}$$

$$2x1 + x2 + s1 = 8 \tag{3.20}$$

$$x2 + s2 = 5 \tag{3.21}$$

$$x1 - x2 + s3 = 4 \tag{3.22}$$

$$x1 \succeq 0; x2 \succeq 0 \tag{3.23}$$

$$s1 \succeq 0; s2 \succeq 0; s3 \succeq 0 \tag{3.24}$$

The standard LP is shown in Table 2.3 below where x1 and x2 are nonbasic or zero variables and s1, s2, and s3 are the basic variables. The starting solution is x1 = 0; x2 = 0; s1 = 8; s2 = 5; s3 = 4 obtained from the RHS column.

Determine the entering and leaving variables. Is the starting solution optimum? No, because Row 0 representing the objective function equation contains nonbasic variables with negative coefficients.

This can also be seen from Figure. In this figure, the current basic solution is shown to be increasing in the direction of the arrow.

Entering Variable: The most negative coefficient in Row 0 is x2. Therefore, the entering variable is x2. This variable must now increase in the direction of the arrow. How far can this increase the objective function? Remember

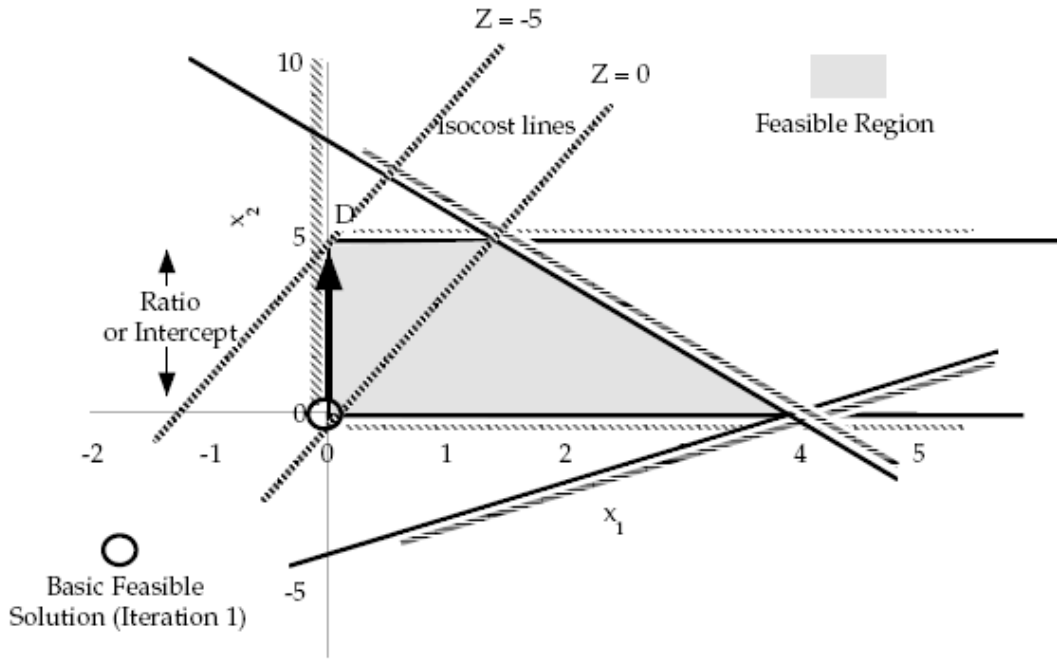


Figure 3.5: Basic solution for simplex example

that the solution has to be in the feasible region. Figure shows that the maximum increase in x2 in the feasible region is given by point D, which is on constraint (2.3). This is also the intercept of this constraint with the y-axis, representing x2. Algebraically, these intercepts are the ratios of the right-hand side of the equations to the corresponding constraint coefficient of x2. We are interested only in the nonnegative ratios, as they represent the direction of increase in x2. This concept is used to decide the leaving variable.

Leaving Variable: The variable corresponding to the smallest nonnegative ratio (5 here) is s2. Hence, the leaving variable is s2.

So, the Pivot Row is Row 2 and Pivot Column is x2.

The two steps of the Gauss–Jordon Row Operation are given below.

The pivot element is underlined in the Table and is 1.

Row Operation:

Pivot: (0, 0, 1, 0, 1, 0, 5)

Row 0: (1, 4,-1, 0, 0, 0, 0)- (-1)(0, 0, 1, 0, 1, 0, 5) = (1, 4, 0, 0, 1, 0, 5)

Row 1: (0, 2, 1, 1, 0, 0, 8)- (1)(0, 0, 1, 0, 1, 0, 5) = (0, 2, 0, 1,-1, 0, 3)

Row 3: (0, 1,-1, 0, 0, 1, 4)- (-1)(0, 0, 1, 0, 1, 0, 5) = (0, 1, 0, 0, 1, 1, 9)

These steps result in the following table (Table).

There is no new entering variable because there are no nonbasic variables with a negative coefficient in row 0. Therefore, we can assume that the solution is reached,

which is given by (from the RHS of each row) x1 = 0; x2 = 5; s1 = 3; s2 = 0; s3 = 9; Z = -5.

Note that at an optimum, all basic variables (x2, s1, s3) have a zero coefficient in Row 0.

3.2 Infeasible Solution

Now consider the same example, and change the right-hand side of Equation2-8 instead of 8. We know that constraint (2) represents the storage capacity and physics tells us that the storage capacity cannot be negative. However, let us see what we get mathematically.

From Figure 2.3, it is seen that the solution is infeasible for this problem. Applying the simplex Method results in Table 2.5 for the first step.

$$
\begin{aligned}
-Z + 4x1 - x2 &= 0 && (3.25)\\
2x1 + x2 + s1 &= -8 \text{ Sorage Constraint} && (3.26)\\
x2 + s2 &= 5 \text{ Availability Constraint} && (3.27)\\
x1 - x2 + s3 &= 4 \text{ Safety Constraint} && (3.28)\\
x1 &\succeq 0; x2 \succeq 0 && (3.29)
\end{aligned}
$$

Standard LP

$$
\begin{aligned}
-Z + 4x1 - x2 &= 0 && (3.30)\\
2x1 + x2 + s1 &= -8 \text{ Sorage Constraint} && (3.31)\\
x2 + s2 &= 5 \text{ Availability Constraint} && (3.32)\\
x1 - x2 + s3 &= 4 \text{ Safety Constraint} && (3.33)
\end{aligned}
$$

The solution to this problem is the same as before: x1 = 0; x2 = 5. However, this solution is not a feasible solution because the slack variable (artificialvariable defined to be always positive) s1 is negative.

3.3 Unbounded Solution

If constraints on storage and availability are removed in the above example, the solution is unbounded, as can be seen in Figure 2.4. This means there are points in the feasible region with arbitrarily large objective function values (for maximization).

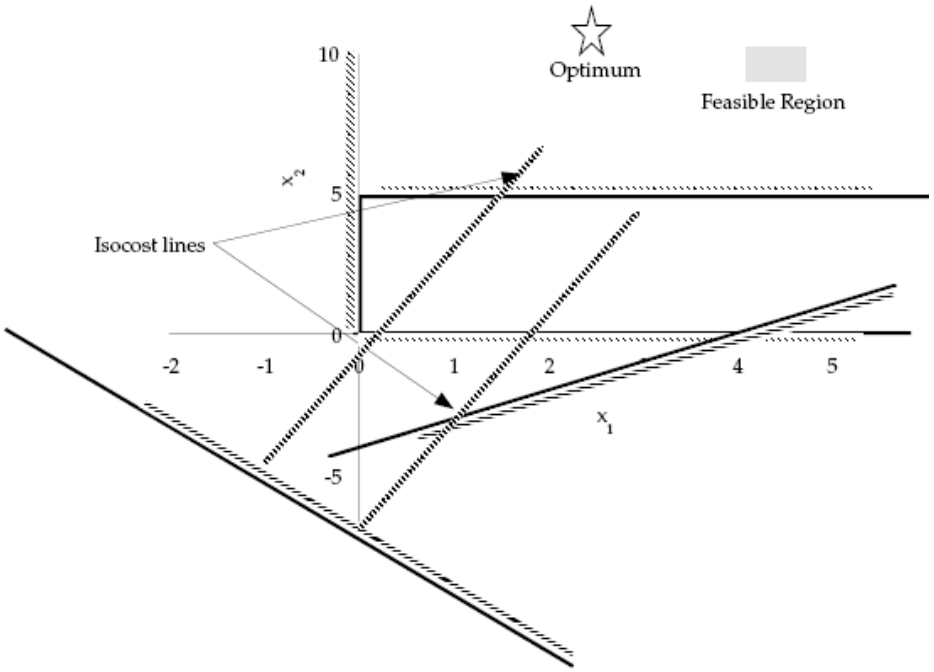


Figure 3.6: Infeasible LP

Row	$-Z$	x_1	x_2	s_1	s_2	s_3	RHS	Basic	Ratio
0	1	4	-1	0	0	0	0	$-Z = 0$	—
1	0	-2	-1	-1	0	0	8	$s_1 = -8$	—
2	0	0	$\underline{1}$	0	1	0	5	$s_2 = 5$	$\underline{5}$
3	0	1	-1	0	0	1	4	$s_3 = 4$	None

Figure 3.7: Initial tableau for infeasible problem

Row	$-Z$	x_1	x_2	s_1	s_2	s_3	RHS	Basic	Ratio
0	1	4	0	0	1	0	5	$-Z = 0$	—
1	0	-2	0	-1	1	0	13	$s_1 = -13$	—
2	0	0	1	0	1	0	5	$x_2 = 5$	—
3	0	1	0	0	1	1	9	$\underline{s_3 = 9}$	—

Figure 3.8: Second iteration tableau for infesible problem

Row	$-Z$	x_1	x_2	s_3	RHS	Basic	Ratio
0	1	4	−1	0	0	$-Z = 0$	—
1	0	1	−1	1	4	$s_3 = 4$	None

Figure 3.9: Simplex tableau for unbounded solution

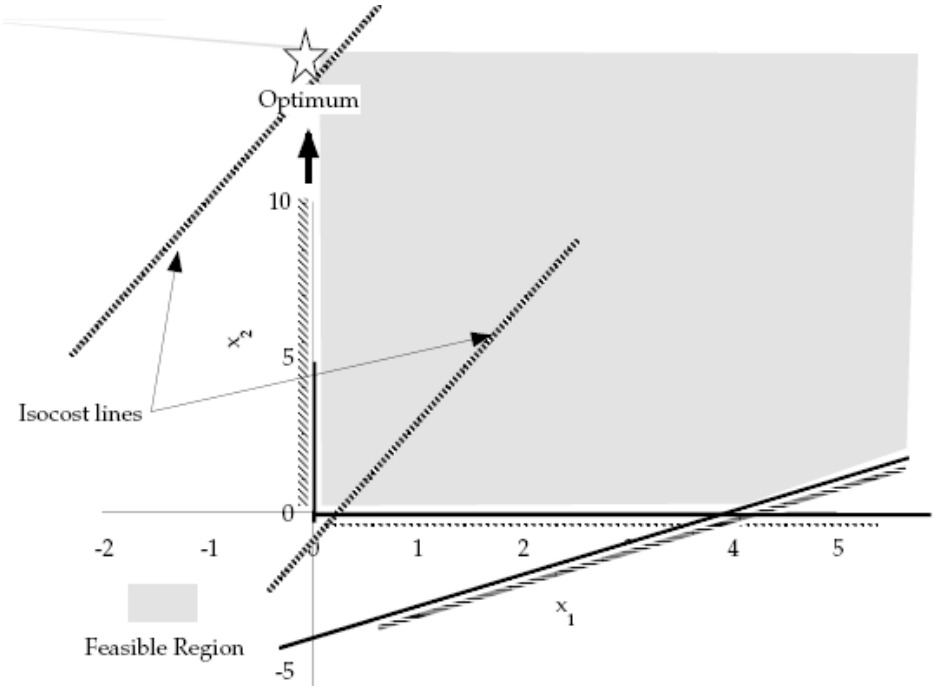


Figure 3.10: Simplex problem for unbounded solution

$$Minimize_{x1,x2}\ Z = 4x1 - x2 \tag{3.34}$$

$$x1 - x2 + s3 = 4 \text{ Safety Constraint} \tag{3.35}$$

$$x1 \succeq 0; x2 \succeq 0 \tag{3.36}$$

The entering variable is x2 as it has the most negative coefficient in row 0. However, there is no leaving variable corresponding to the binding constraint (the smallest non-negative ratio or intercept). That means x2 can take as high a value as possible. This is also apparent in the graphical solution shown in Figure The LP is unbounded when (for a maximization problem) a nonbasic variable with a negative coefficient in row 0 has a nonpositive coefficient in each constraint, as shown in the table·

3.4 Multiple Solutions

In the following example, the cost of X1 is assumed to be negligible as compared to the credit of X2. This LP has infinite solutions given by the isocost line (x2 = 5) . The simplex method generally finds one solution at a time. Special methods such as goal programming or multiobjective optimization can be used to find these solutions.

3.4.1 Matlab code for Linear Programming (LP)

The linear programming (LP) scheme implemented by the MATLAB built-in routine

"[xo,fo] = linprog(f,A,b,Aeq,Beq,l,u,x0,options)"

is designed to solve an LP problem, which is a constrained minimization problem as follows.

$$Minf(x) = f^T x \tag{3.37}$$

subject to

$$Ax \leq b, Aeqx = beq, andl \leq x \leq u \tag{3.38}$$

%nm733 to solve a Linear Programming problem.
% Min f*x=-3*x(1)-2*x(2) s.t. Ax <= b, Aeq = beq and l <= x <= u
x0 = [0 0]; %initial point
f = [-3 -2]; %the coefficient vector of the objective function
A = [3 4; 2 1]; b = [7; 3]; %the inequality constraint Ax <= b
Aeq = [-3 2]; beq = 2; %the equality constraint Aeq*x = beq
l = [0 0]; u = [10 10]; %lower/upper bound l <= x <= u
[xo_lp,fo_lp] = linprog(f,A,b,Aeq,beq,l,u)
cons_satisfied = [A; Aeq]*xo_lp-[b; beq] %how constraints are satisfied
f733o=inline('-3*x(1)-2*x(2)', 'x');
[xo_con,fo_con] = fmincon(f733o,x0,A,b,Aeq,beq,l,u)

It produces the solution (column) vector xo and the minimized value of the objective function f (xo) as its first and second output arguments xo and fo, where the objective function and the constraints excluding the constant term are linear in terms of the independent (decision) variables. It works for such linear optimization problems more efficiently than the general constrained optimization routine "fmincon()".

Chapter 4

Nonlinear Programming

In nonlinear programming (NLP) problems, either the objective function, the constraints, or both the objective and the constraints are nonlinear, as shown below in Example.

Consider a simple isoperimetric problem described in Chapter

1. Given the perimeter (16 cm) of a rectangle, construct the rectangle with maximum area. To be consistent with the LP formulations of the inequalities seen earlier, assume that the perimeter of 16 cm is an upper bound to the real perimeter.

Solution: Let x1 and x2 be the two sides of this rectangle. Then the problem can be formulated as a nonlinear programming problem with the nonlinear objective function and the linear inequality constraints given below:

Maximize Z = x1 x x2

x1, x2

subject to

2x1 + 2x2 ■ 16 Perimeter Constraint

x1 ■ 0; x2 ■ 0

Let us start plotting the constraints and the iso-objective (equal-area) contours in Figure 3.1. As stated earlier in the figure, the three inequalities are represented by the region on the other side of the hatched lines. The objective function lines are represented as dashed contours. The optimal solution is at x1 = 4 cm; x2 = 4 cm. Unlike LP, the NLP solution is not lying at the vertex of the feasible region, which is the basis of the simplex method.

The above example demonstrates that NLP problems are different from LP problems because:

• An NLP solution need not be a corner point.

• An NLP solution need not be on the boundary (although in this example it is on the boundary) of the feasible region. It is obvious that one cannot use the simplex for solving an NLP. For an NLP solution, it is necessary to look at the relationship of the objective

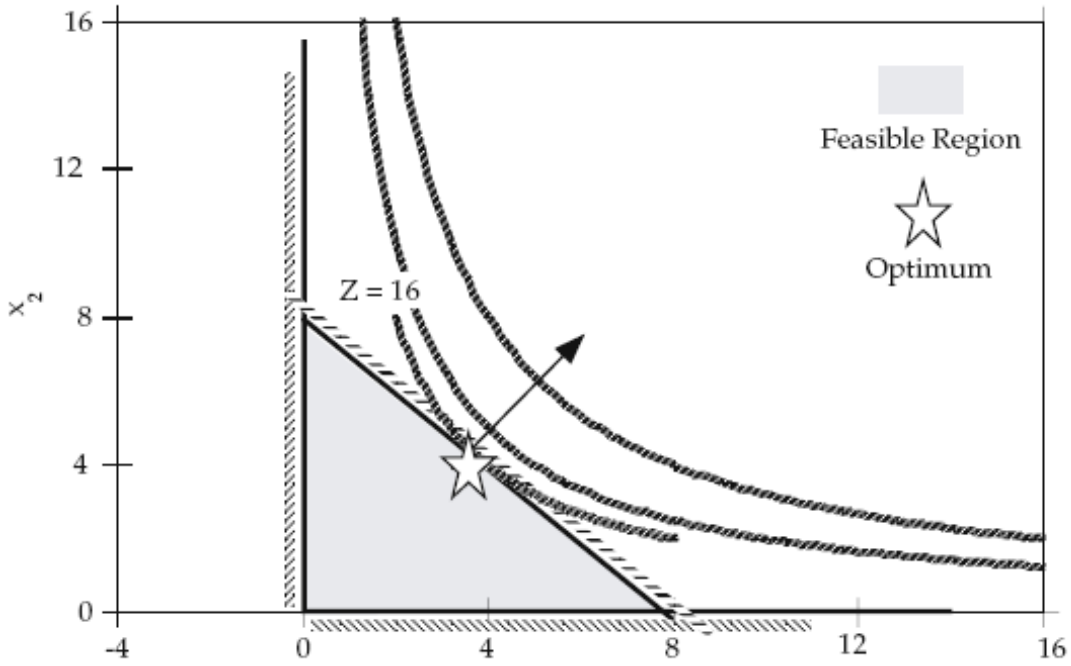


Figure 4.1: Nonlinear programming contour plot

function to each decision variable. Consider the previous example. Let us convert the problem into a onedimensional problem by assuming constraint (isoperimetric constraint) as an equality. One can eliminate x2 by substituting the value of x2 in terms of x1 using constraint.

$$\min_{x1,x2} Z = 8x_1 - x_1^2 \tag{4.1}$$

$$x1 \succeq 0 \tag{4.2}$$

Figure shows the graph of the objective function versus the single decision variable x1. In Figure , the objective function has the highest value (maximum) at x1 = 4. At this point in the figure, the x-axis is tangent to the objective

function curve, and the slope dZ/dx1 is zero. This is the first condition that is used in deciding the extremum point of a function in an NLP setting. Is this a minimum or a maximum? Let us see what happens if we convert this maximization problem into a minimization problem with -Z as the objective function.

$$\min_{x1} -Z = 8x_1 - x_1^2 \tag{4.3}$$

$$x1 \succeq 0 \tag{4.4}$$

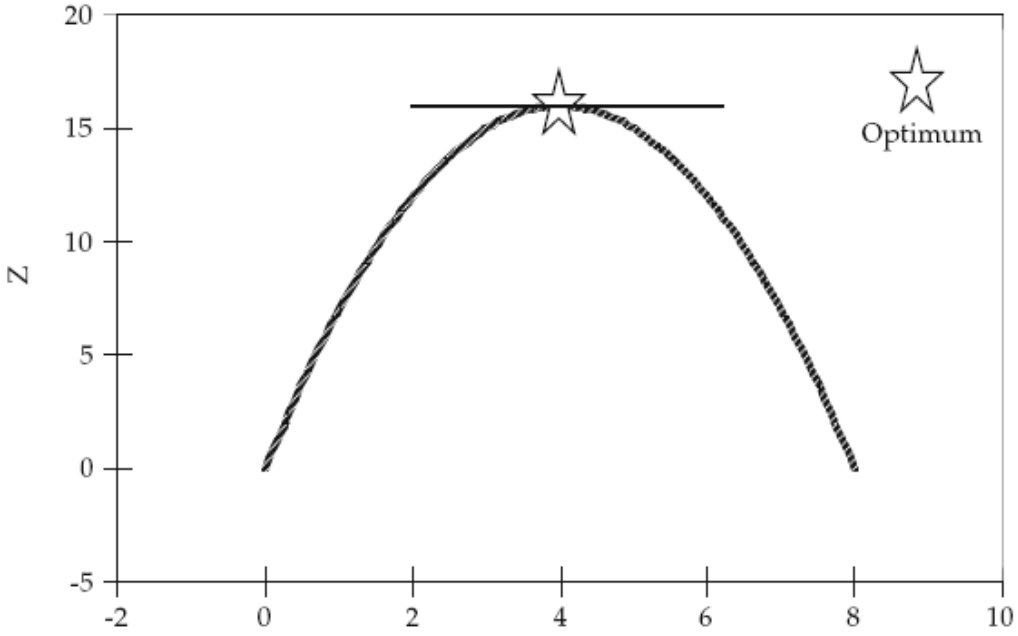


Figure 4.2: Nonlinear program graphical representation

Figure shows that -Z has the lowest value at the same point, x1 = 4. At this point in both figures, the x-axis is tangent to the objective function curve, and slope dZ/dx1 is zero. It is obvious that for both the maximum and minimum points, the necessary condition is the same. What differentiates a minimum from a maximum is whether the slope is increasing or decreasing around the extremum point. In Figure 3.2, the slope is decreasing as you move away from x1 = 4, showing that the solution is a maximum. On the other hand, in Figure 3.3 the slope is increasing, resulting in a minimum.

Whether the slope is increasing or decreasing (sign of the second derivative) provides a sufficient condition for the optimal solution to an NLP.

Many times there will be more than one minia existing. For the case shown in the figure the number of minim are two more over one being better than the other.This is another case in which an NLP differs from an LP, as

• In LP, a local optimum (the point is better than any "adjacent" point) is a global (best of all the feasible points) optimum. With NLP, a solution can be a local minimum.

• For some problems, one can obtain a global optimum. For example, – Figure shows a global maximum of a concave function.

– Figure presents a global minimum of a convex function. What is the relation between the convexity or concavity of a function and

its optimum point? The following section describes convex and concave functions and their relation to the NLP solution.

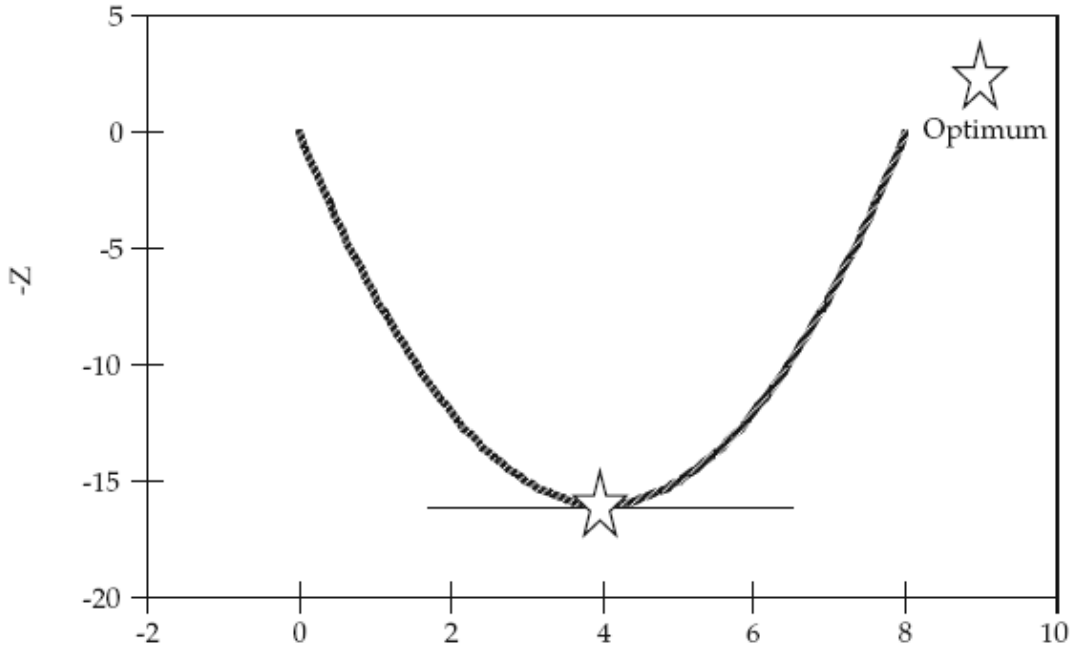


Figure 4.3: Nonlinear programming minimum

4.1 Convex and Concave Functions

A set of points S is a convex set if the line segment joining any two points in the space S is wholly contained in S. In Figure 3.5, a and b are convex sets, but c is not a convex set.

Mathematically, S is a convex set if, for any two vectors x1 and x2 in S, the vector x = ■x1 +(1-■)x2 is also in S for any number ■ between 0 and 1. Therefore, a function f(x) is said to be strictly convex if, for any two distinct points x1 and x2, the following equation applies.

f(■x1 + (1 - ■)x2) < ■f(x1) + (1 - ■)f(x2) (3.7)

Figure 3.6a describes Equation (3.7), which defines a convex function. This convex function (Figure 3.6a) has a single minimum, whereas the nonconvex function can have multiple minima. Conversely, a function f(x) is strictly concave if -f(x) is strictly convex.

As stated earlier, concave function has a single maximum.

Therefore, to obtain a global optimum in NLP, the following conditions apply.

• Maximization: The objective function should be concave and the solution space should be a convex set .

• Minimization: The objective function should be convex and the solution space should be a convex set .

Note that every global optimum is a local optimum, but the converse is not true. The set of all feasible solutions to a linear programming problem is a convex set. Therefore,

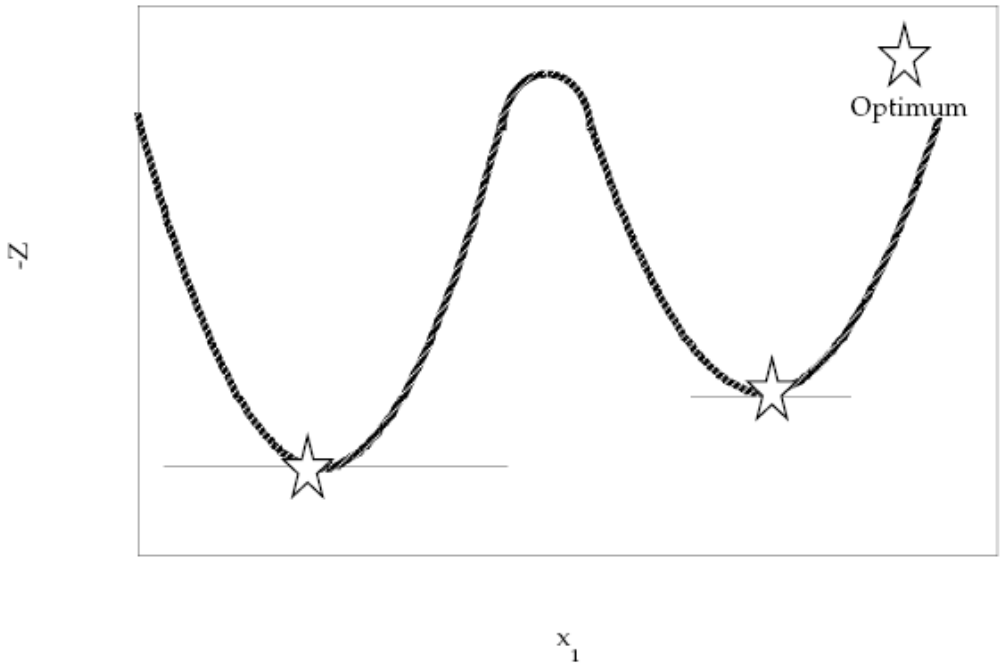


Figure 4.4: Nonlinear programming multiple minimum

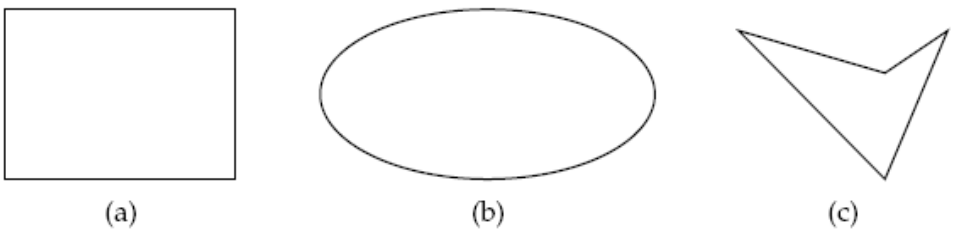


Figure 4.5: Examples of convex and nonconvex sets

a linear programming optimum is a global optimum. It is clear that the NLP solution depends on the objective function and the solution space defined by the constraints. The following sections describe the unconstrained and constrained NLP, and the necessary and sufficient conditions for obtaining the optimum for these problems.

Chapter 5

Discrete Optimization

Discrete optimization problems involve discrete decision variables as shown below in Example Consider the isoperimetric problem solved to be an NLP. This problem is stated in terms of a rectangle. Suppose we have a choice among a rectangle, a hexagon, and an ellipse, as shown in Figure

Draw the feasible space when the perimeter is fixed at 16 cm and the objective is to maximize the area.

Solution: The decision space in this case is represented by the points corresponding to different shapes and sizes as shown Discrete optimization problems can be classified as integer programming (IP) problems, mixed integer linear programming (MILP), and mixed integer

nonlinear programming (MINLP) problems. Now let us look at the decision variables associated with this isoperimetric problem. We need to decide which shape and what dimensions to choose. As seen earlier, the dimensions of a particular figure represent continuous decisions in a real domain, whereas

selecting a shape is a discrete decision. This is an MINLP as it contains both continuous (e.g., length) and discrete decision variables (e.g., shape), and the objective function (area) is nonlinear. For representing discrete decisions associated with each shape, one

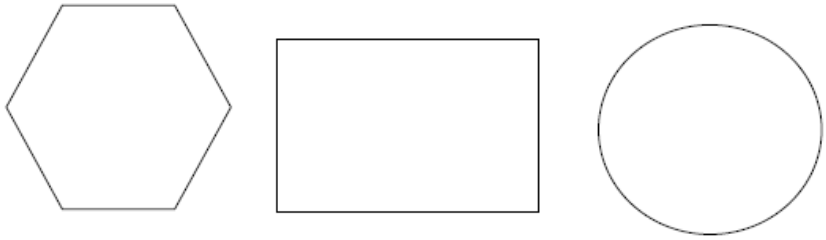


Figure 5.1: Isoperimetric problem discrete decisions

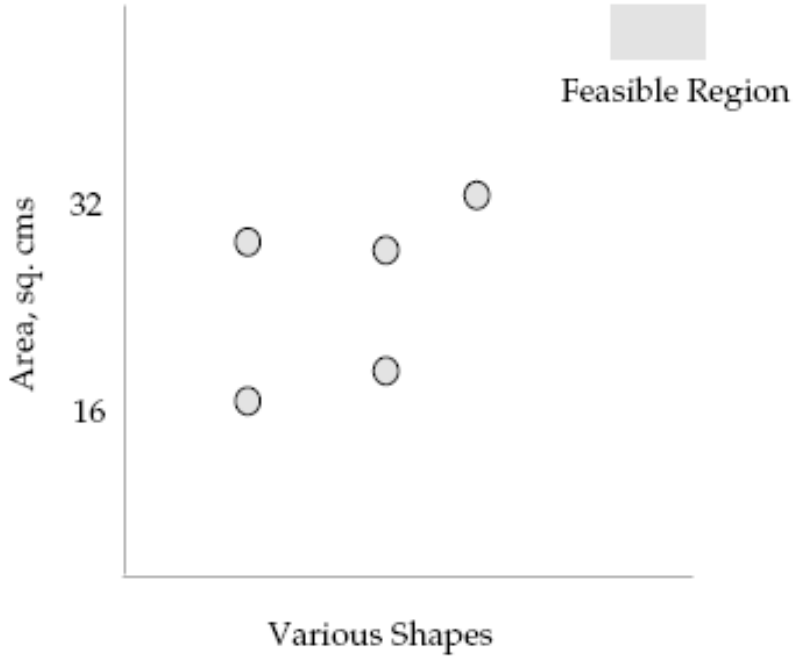


Figure 5.2: Feasible space for discrete isoperimetric problem

can assign an integer for each shape or a binary variable having values of 0 and 1 (1 corresponding to yes and 0 to no). The binary variable representation is used in traditional mathematical programming algorithms for solving problems involving discrete decision variables.

However, probabilistic methods such as simulated annealing and genetic algorithms which are based on analogies to a physical process such as the annealing of metals or to a natural process such as genetic evolution, may prefer to use different integers assigned to different decisions.

Representation of the discrete decision space plays an important role in selecting a particular algorithm to solve the discrete optimization problem. The following section presents the two different representations commonly used in discrete optimization.

5.1 Tree and Network Representation

Discrete decisions can be represented using a tree representation or a network representation. Network representation avoids duplication and each node corresponds to a unique decision. This representation is useful when one is using methods like discrete dynamic programming. Another advantage of the network representation is that an IP problem that can be represented.

appropriately using the network framework can be solved as an LP. Examples of

Separator	Cost
A/BCD	50
AB/CD	170
ABC/D	110
A/BC	40
AB/C	69
B/CD	228
BC/D	40
A/B	144
B/C	50
C/D	329

Figure 5.3: Cost of seperations 1000 Rs/year

network models include transportation of supply to

satisfy a demand, flow of wealth, assigning jobs to machines, and project management. The tree representation shows clear paths to final decisions; however, it involves duplication. The tree representation is suitable when the discrete decisions are represented separately, as in the Branch-and-bound method.

This method is more popular for IP than the discrete dynamic programming method in the mathematical programming literature due to its easy implementation and generalizability. The following example from Hendry and Hughes (1972) illustrates the two representations.

Example 1 *Given a mixture of four chemicals A, B, C, D for which differenttechnologies are used to separate the mixture of pure components. The costof each technology is given in Table 4.1 below. Formulate the problem as an optimization problem with tree and network representations.*

Solution: Figure 4.3 shows the decision tree for this problem. In this representation, we have multiple representations of some of the separation options. For example, the binary separators A/B, B/C, C/D appear twice in the terminal nodes. We can avoid this duplication by using the network representation shown in Figure . In this representation, we have combined the branches that lead to the same binary separators. The network representation has 10 nodes, and the tree representation has 13 nodes. The optimization problem is to find the path that will separate the mixture into pure components for a minimum cost. From the two representations, it is very clear that the decisions involved here are all discrete decisions. This is a pure integer programming problem. The mathematical programming method commonly used to solve this problem is the Branch-and-bound method. This method is described in the next section.

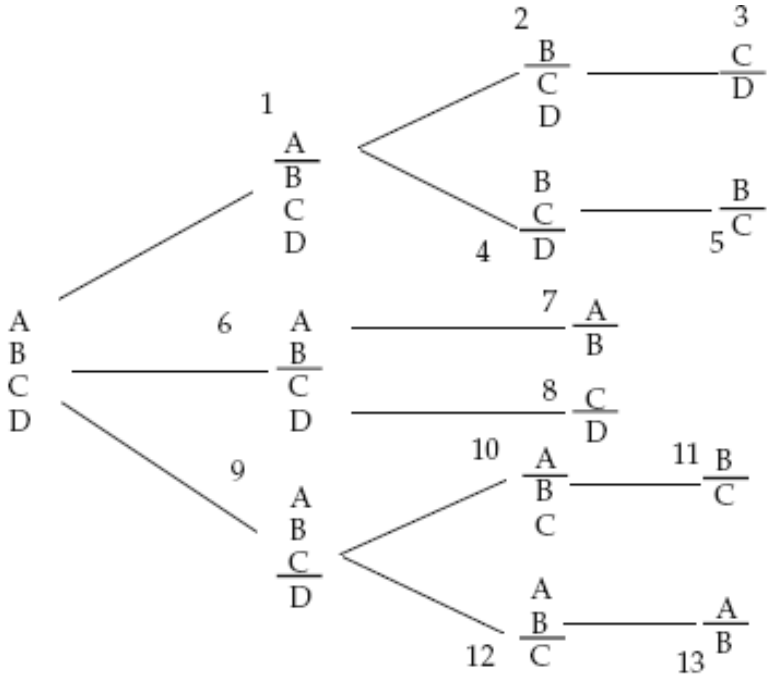


Figure 5.4: Tree representation

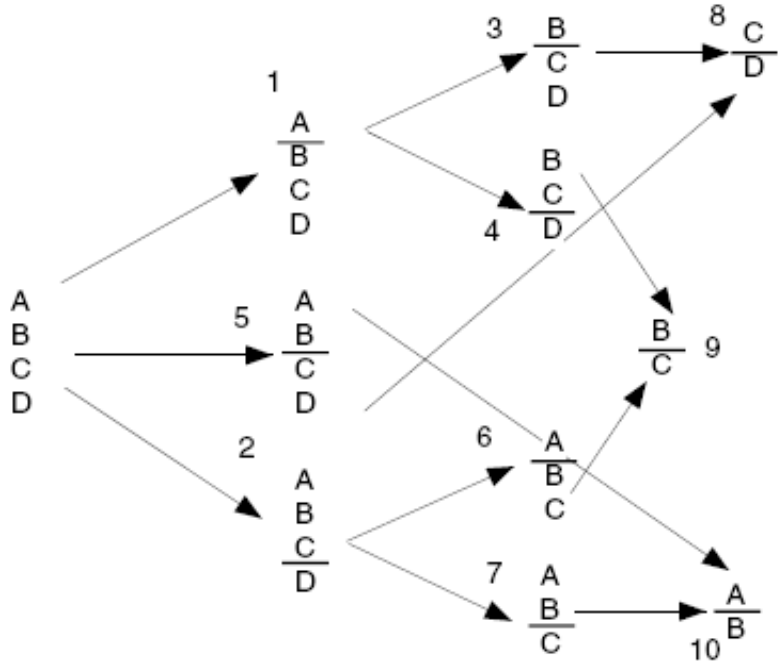


Figure 5.5: Network representation

5.2 Branch-and-Bound for IP

Having developed the representation, the question is how to search for the optimum. One can go through the complete enumeration, but that would involve evaluating each node of the tree. The intelligent way is to reduce the search space by implicit enumeration and evaluate as few nodes as possible.

Consider the above example of separation sequencing. The objective is to minimize the cost of separation. If one looks at the nodes for each branch, there are an initial node, intermediate nodes, and a terminal node. Each node is the sum of the costs of all earlier nodes in that branch. Because this cost increases monotonically as we progress through the initial, intermediate, and final nodes, we can define the upper bound and lower bounds for each branch.

• The cost accumulated at any intermediate node is a lower bound to the cost of any successor nodes, as the successor node is bound to incur additional cost.

• For a terminal node, the total cost provides an upper bound to the original problem because a terminal node represents a solution that may or may not be optimal. The above two heuristics allow us to prune the tree for cost minimization. If the cost at the current node is greater than or equal to the upper bound defined earlier either from one of the prior branches or known to us from experience, then we don't need to go further in that branch. These are the two common ways to prune the tree based on the order in which the nodes are enumerated:

• Depth-first: Here, we successively perform one branching on the most recently created node. When no nodes can be expanded, we backtrack to a node whose successor nodes have not been examined.

• Breadth-first: Here, we select the node with the lowest cost and expand all its successor nodes. The following example illustrates these two strategies for the problem specified in Example previously. Find the lowest cost separation sequence for the problem specified in PreviousExample using the depth-first and breadth-first Branch-and-bound strategies.

Solution: Consider the tree representation shown in Figure for this problem. First, let's examine the depth-first strategy, as shown in Figure and enumerated below.

• Branch from Root Node to Node 1: Sequence Cost = 50.

• Branch from Node 1 to Node 2: Sequence Cost = 50 + 228 = 278.

• Branch from Node 2 to Node 3: Sequence Cost = 278 + 329 = 607.

– Because Node 3 is terminal, current upper bound = 607.

– Current best sequence is (1, 2, 3).

• Backtrack to Node 2.

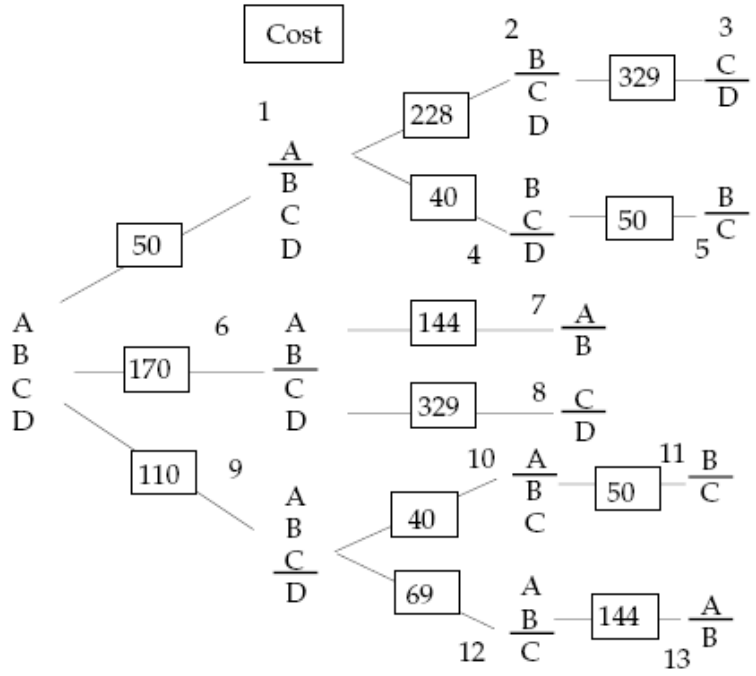


Figure 5.6: Tree representation and cost diagram

• Backtrack to Node 1

Branch from Node 1 to Node 4: Sequence Cost = 50 + 40 = 90 < 607.

• Branch from Node 4 to Node 5: Sequence Cost = 90 + 50 = 140 < 607.

– Because Node 5 is terminal and 140 < 607, current upper bound = 140.

– Current best sequence is (1, 4, 5).

• Backtrack to Node 4.

• Backtrack to Node 1.

• Backtrack to Root Node.

• Branch from Root Node to Node 6: Sequence Cost = 170.

– Because 170 > 140, prune Node 6.

– Current best sequence is still (1, 4, 5).

• Backtrack to Root Node.

• Branch from Root Node to Node 9: Sequence Cost = 110.

– Branch from Node 9 to Node 10: Sequence Cost = 110 + 40 = 150.

– Branch from Node 9 to Node 12: Sequence Cost = 110 + 69 = 179.

– Because 150 > 140, prune Node 10.

– Because 179 > 140, prune Node 12.

– Current best sequence is still (1, 4, 5).

• Backtrack to Root Node.

• Because all the branches from the Root Node have been examined, stop.

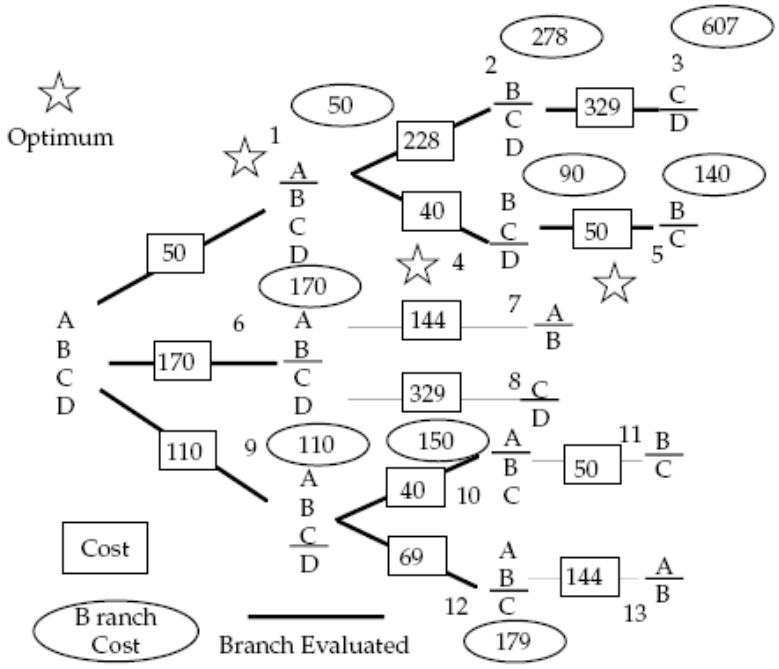


Figure 5.7: Deapth first enumeration

• Optimal sequence (1, 4, 5), Minimum Cost = 140.

Note that with the depth-first strategy, we examined 9 nodes out of 13 that we have in the tree. If the separator costs had been a function of continuous decision variables, then we would have had to solve either an LP or an NLP at each node, depending on the problem type. This is the principle behind the depth-first Branch-and-bound strategy. The breadth-first strategy enumeration is shown in Figure . The steps are elaborated below.

• Branch from root node to:

– Node 1: Sequence cost = 50.

– Node 6: Sequence cost = 170.

– Node 9: Sequence cost = 110.

• Select Node 1 because it has the lowest cost.

• Branch Node 1 to:

– Node 2: Sequence Cost = 50 + 228 = 278.

– Node 4: Sequence Cost = 50 + 40 = 90.

• Select Node 4 because it has the lowest cost among 6, 9, 2, 4.

• Branch Node 4 to:

– Node 5: Sequence Cost = 90 + 50 = 140.

• Because Node 5 is terminal, current best upper bound = 140 with the current best sequence (1, 4, 5).

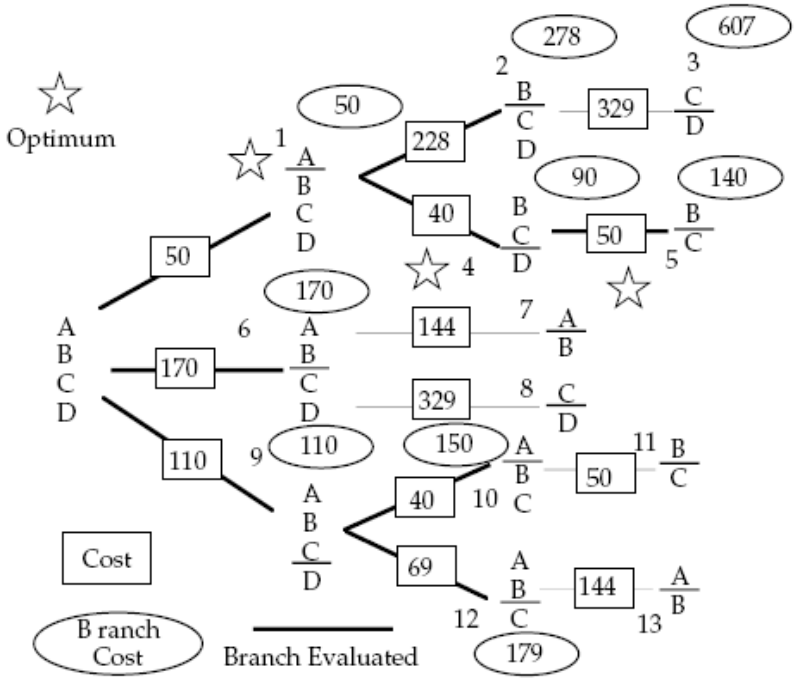


Figure 5.8: Breadth first enumeration

• Select Node 9 because it has the lowest cost among 6, 9, 2, 5.

• Branch Node 9 to:

– Node 10: Sequence Cost = 110 + 40 = 150.

– Node 12: Sequence Cost = 110 + 69 = 179.

From all the available nodes 6, 2, 5, 10, and 12, Node 5 has the lowest cost, so stop.

• Optimal Sequence (1, 4, 5), Minimum Cost = 140. Note that with the breadth-first strategy, we only had to examine 8 nodes

out of 13 nodes in the tree, one node less than the depth-first strategy. In general, the breadth-first strategy requires the examination of fewer nodes and no backtracking. However, depth-first requires less storage of nodes because the maximum number of nodes to be stored at any point is the number of levels in the tree. For this reason, the depth-first strategy is commonly used. Also, this strategy has a tendency to find the optimal solution earlier than the breadth-first strategy. For example, in Example, we had reached the optimal solution in the first few steps using the depth-first strategy (seventh step, with five nodes examined)

Chapter 6

Integrated Planning and Scheduling of processes

6.1 Introduction

In this chapter, we address each part of the manufacturing business hierarchy, and explain how optimization and modeling are key tools that help link the components together. Also introduce the concept of scheduling and resent developments related to schduling of batch process.

6.2 Plant optimization hierarchy

Figure (??)shows the relevant levels for the process industries in the optimization hierarchy for business manufacturing. At all levels the use of optimization techniques can be pervasive although specific techniques are not explicitly listed in the specific activities shown in the figure. In Figure (??)the key information sources for the plant decision hierarchy for operations are the enterprise data, consisting of ' commercial and financial information, and plant data, usually containing the values of a large number of process variables. The critical linkage between models and optimization in all of the five levels is illustrated in Figure(??). The first level (planning) sets production goals that meet supply and logistics constraints, and scheduling (layer 2) addresses time-varying capacity and staffing utilization decisions. The term supply chain refers to the links in a web of relationships involving materials acquisition, retailing (sales), distribution, transportation, and manufacturing with suppliers. Planning and scheduling usually take place over relatively long time frames and tend to be loosely coupled to the information flow and analysis that occur at lower levels in the hierarchy. The time scale for decision making at the highest level (planning) may be on the order of months, whereas at the lowest

level (e.g., process monitoring) the interaction with the process may be in fractions of a second.

Plantwide management and optimization at level 3 coordinates the network of process units and provides cost-effective setpoints via real-time optimization. The unit management and control level includes process control [e.g., optimal tuning of proportional-integral-derivative (PID) controllers], emergency response, and diagnosis, whereas level 5 (process monitoring and analysis) provides data acquisition and online angysis and reconciliation functions as well as fault detection. Ideally, bidirectional communication occurs between levels, with higher levels setting goals for lower levels and the lower levels communicating constraints and performance information to the higher levels. Data are collected directly at all levels in the enterprise. In practice the decision flow tends to be top down, invariably resulting in mismatches between goals and their realization and the consequent accumulation of inventory. Other more deleterious effects include reduction of processing capacity, off-specification products, and failure to meet scheduled deliveries.

Over the past 30 years, business automation systems and plant automation systems have developed along different paths, particularly in the way data are acquired, managed, and stored. Process management and control systems normally use the same databases obtained from various online measurements of the state of the plant. Each level in Figure (??p1) may have its own manually entered database, however, some of which are very large, but web-based data interchange will facilitate standard practices in the future.

Table (??p1) lists the kinds of models and objective functions used in the computer-integrated manufacturing (CIM) hierarchy. These models are used to make decisions that reduce product costs, improve product quality, or reduce time to market (or cycle time). Note that models employed can be classified as steady state or dynamic, discrete or continuous, physical or empirical, linear or nonlinear, and with single or multiple periods. The models used at different levels are not normally derived from a single model source, and as a result inconsistencies in the model can arise. The chemical processing industry is, however, moving in the direction of unifying the modeling approaches so that the models employed are consistent and robust, as implied in Figure (??p1). Objective functions can be economically based or noneconomic, such as least squares.

Planning and Scheduling

Bryant (1993) states that planning is concerned with broad classes of products and the provision of adequate manufacturing capacity. In contrast, scheduling focuses on details of material flow, manufacturing, and production, but still may be concerned with offline planning. Reactive scheduling refers to real-time scheduling and the handling of unplanned changes in demands or resources. The term enterprise resource planning (ERP) is used today, replacing the term manufacturing resources planning (MRP); ERP may or

may not explicitly include planning and scheduling, depending on the industry. Planning and scheduling are viewed as distinct levels in the manufacturing hierarchy as shown in Figure (??p1), but often a fair amount of overlap exists in the two problem statements, as discussed later on. The time scale can often be the determining factor in whether a given problem is a planning or scheduling one: planning is typified by a time horizon of months or weeks, whereas scheduling tends to be of shorter duration, that is, weeks, days, or hours, depending on the cycle time from raw materials to final product. Bryant distinguishes among system operations planning, plant operations planning, and plant scheduling, using the tasks listed in Table (??t2). At the systems operations planning level traditional multiperiod, multilocation linear programming problems must be solved, whereas at the plant operations level, nonlinear multiperiod models may be used, with variable time lengths that can be optimized as well (Lasdon and Baker, 1986).

Baker (1993) outlined the planning and scheduling activities in a refinery as follows:

1. The corporate operations planning model sets target levels and prices for inter-refinery transfers, crude and product allocations to each refinery, production targets, and inventory targets for the end of each refinery model's time horizon.

2. In plant operations planning each refinery model produces target operating conditions, stream allocations, and blends across the whole refinery, which determines.

(a) optimal operating conditions, flows, blend recipes, and inventories; and (b) costs, cost limits, and marginal values to the scheduling and real-time optimization (RTO) models.

3. The scheduling models for each refinery convert the preceding information into detailed unit-level directives that provide day-by-day operating conditions or-set points.

Supply chain management poses difficult decision-making problems because of its wide ranging temporal and geographical scales, and it calls for greater responsiveness because of changing market factors, customer requirements, and plant availability. Successful supply chain management must anticipate customer requirements, commit to customer orders, procure new materials, allocate production

capacity, schedule production, and schedule delivery. According to Bryant (1993), the costs associated with supply chain issues represent about 10 percent of the sales value of domestically delivered products, and as much as 40 percent internationally.

Managing the supply chain effectively involves not only the manufacturers, but also their trading partners: customers, suppliers, warehousers, terminal operators, and transportation carriers (air, rail, water, land).

In most supply chains each warehouse is typically controlled according to some local law such as a safety stock level or replenishment rule. This local control can cause buildup of inventory at a specific point in the system and thus propagate disturbances over the

time frame of days to months (which is analogous to disturbances in the range of minutes or hours that occur at the production control level). Short-term changes that can upset the system include those that are "selfinflicted" (price changes, promotions, etc.) or effects of weather or other cyclical consumer patterns. Accurate demand forecasting is critical to keeping the supply chain network functioning close to its optimum when the produce-to-inventory approach is used.

6.3 Planning

A simplified and idealized version of the components involved in the planning step, that is, the components of the supply chain incorporates. S possible suppliers provide raw materials to each of the M manufacturing plants. These plants manufacture a given product that may be stored or warehoused in W facilities (or may not be stored at all), and these in turn are delivered to C different customers. The nature of the problem depends on whether the products are made to order or made to inventory; made to order fulfills a specific customer order, whereas made to inventory is oriented to the requirements of the general market demand., with material balance conditions satisfied between suppliers, factories, warehouses, and customers (equality constraints). Inequality constraints would include individual line capacities in each manufacturing plant, total factory capacity, warehouse storage limits, supplier limits, and customer demand. Cost factors include variable manufacturing costs, cost of warehousing, supplier prices, transportation costs (between each sector), and variable customer pricing, which may be volume and quality-dependent. A practical problem may involve as many as 100,000 variables and can be solved using mixed-integer linear programming (MILP).

Most international oil companies that operate multiple refineries analyze the refinery optimization problem over several time periods (e.g., 3 months). This is because many crudes must be purchased at least 3 months in advance due to transportation requirements (e.g., the need to use tankers to transport oil from the Middle East). These crudes also have different grades and properties, which must be factored into the product slate for the refinery. So the multitime period consideration is driven more by supply and demand than by inventory limits (which are typically less than 5 days). The LP models may be run on a weekly basis to handle such items as equipment changes and maintenance, short-term supply issues (and delays in shipments due to weather problems or unloading difficulties), and changes in demand (4 weeks within a 1-month period). Product properties such as the Reid vapor pressure must be changed between summer and winter months to meet environmental restrictions on gasoline properties. See Pike (1986) for a detailed LP refinery example that treats quality specifications and physical properties by using

product blending, a dimension that is relevant for companies with varied crude supplies and product requirements.

6.4 Scheduling

Information processing in production scheduling is essentially the same as in planning. Both plants and individual process equipment take orders and make products. or a plant, the customer is usually external, but for a process (or "work cell" in discrete manufacturing parlance), the order comes from inside the plant or factory. In a plant, the final product can be sold to an external customer; for a process, the product delivered is an intermediate or partially finished product that goes on to the next stage of processing (internal customer).

Two philosophies are used to solve production scheduling problems (Puigjaner and Espura, 1998):

1. The top-down approach, which defines appropriate hierarchical coordination mechanisms between the'different decision levels and decision structures at each level. These structures force constraints on lower operating levels and require heuristic decision rules for each task. Although this approach reduces the size and complexity of scheduling problems, it potentially introduces coordination problems.

2. The bottom-up approach, which develops detailed plant simulation and optimization models, optimizes them, and translates the results from the simulations and optimization into practical operating heuristics. This approach often leads to large models with many variables and equations that are difficult to solve quickly using rigorous optimization algorithms.

Table (??t3) categorizes the typical problem statement for the manufacturing scheduling and planning problem. In a batch campaign or run, comprising smaller runs called lots, several batches of product may be produced using the same recipe. To optimize the production process, you need to determine

1. The recipe that satisfies product quality requirements.

2. The production rates needed to fulfill the timing requirements, including any precedence constraints.

3. Operating variables for plant equipment that are subject to constraints.

4. Availability of raw material inventories.

5. Availability of product storage.

6. The run schedule.

7. Penalties on completing a production step too soon or too late.

Example 2 *MULTIPRODUCT BATCH PLANT SCHEDULING: Batch operations such*

as drying, mixing, distillation, and reaction are widely used in producing food, pharmaceuticals, and specialty products (e.g., polymers). Scheduling of operations as described in Table 16.3 is crucial in such plants. A principal feature of batch plants (Ku and Karimi, 1987) is the production of multiple products using the same set of equipment. Good industrial case studies of plant scheduling include those by Bunch et al. (1998), McDonald (1998), and Schulz et al. (1998). For example, Schulz et al. described a polymer plant that involved four process steps (preparation, reaction, mixing, and finishing) using different equipment in each step. When products are similar in nature, they require the same processing steps and hence pass through the same series of processing units; often the batches are produced sequentially. Such plants are called multiproduct plants. Because of different processing time requirements, the total time required to produce a set of batches (also called the makespan or cycle time) depends on the sequence in whidh they are produced. To maximize plant productivity, the batches should be produced in a sequence that minimizes the makespan. The plant schedule corresponding to such a sequence can then be represented graphically in the form of a Gantt chart (see the following discussion and Figure E16.2b). The Gantt chart provides a timetable of plant operations showing which products are produced by which units and at what times. In this example we consider four products @I, p2, p3, p4) that are to be produced as a series of batches in a multiproduct plant consisting of three batch reactors in series (Ku and Karirni, 1992); . The processing times for each batch reactor and each product are given in Table E16.2. Assume that no intermediate storage is available between the processing units. If a product finishes its processing on unit k and unit k + 1 is not free because it is still processing a previous product, then the completed product must be kept in unit k, until unit k + 1 becomes free. As an example, product pl must be held in unit 1 until unit 2 finishes processing p3. When a product finishes processing on the last unit, it is sent immediately to product storage. Assume that the times required to transfer products from one unit to another are negligible compared with the processing times. The problem for this example is to determine the time sequence for producing the four products so as to minimize the makespan. Assume that all the units are initiallyempty (initialized) at time zero and the manufacture of any product can be delayed an arbitrary amount of time by holding it in the previous unit.

Solution 3 *Let N be the number of products and M be the number of units in the plant.Let q„ (called completion time) be the "clock time at which the jth product in the sequence leaves unit k after completion of its processing, and let rLkb e the time required to process the jth product in the sequence on unit k (See Table E16.2). The first product goes into unit 1 at time zero, so ClYo= 0. The index j in T ˜an,d˜C jrk denotes the position of a product in the sequence. Hence CN„ is the time at which the last product leaves the last unit and is the makespan to be minimized. Next, we derive the set of constraints (Ku*

and Karimi, 1988; 1990) that interrelate the Cj,k. First, the jth product in the sequence cannot leave unit k until it is processed, and in order to be processed on unit k, it must have left unit k - 1. Therefore the clock time at which it leaves unit k (i.e., q,+) m ust be equal to or after the time at which it leaves unit k - 1 plus the processing time in k. Thus the first set of constraints in the formulation is Similarly, the jth product cannot leave unit k until product (j - 1) has been processed and transferred: Set C„ = 0. Finally the jth product in the sequence cannot leave unit k until the downsbeam unit k + 1 is free [i.e., product (j - 1) has left]. Therefore Although Equations (a)-(c) represent the complete set of constraints, some of them are redundant. From Equation (a) Cj„ r Cj,k- + T ˜fo,r˜k 1 2 . But from Equation (c), Cj,k-l 2 Cj- l,k, hence Cj,k 1 Cj- l,k + ri,k for k = 2, M. In essence, Equations (a) and (c) imply Equations (b) for k = 2, M, so Equations (b) for k = 2, M are redundant. Having derived the constraints for completion times, we next determine the sequence of operations. In contrast to the CjPkt, he decision variables here are discrete (binary). Define Xij as follows. Xi,. = 1 if product i (product with label pi) is in slot j of the sequence, otherwise it is zero. So X3„ = 1 means that product p3 is second in the production sequence, and X3, = 0 means that it is not in the second position. The overall integer constraint is Similarly every product should occupy only one slot in the sequence: The Xij that satisfy Equations (d) and (e) always give a meaningful sequence. Now we must determine the clock times ti„ for any given set of Xi,j. If product pi is in slot j, then tj,km˜˜tbe7i,kandX= i1,j a ndXi,l = Xi„ = . . . = Xi ,.˜- 1 = X˜.. +=l . . . = xi,N= 0, therefore we can use XiFj to pick the right processing time representing $„ by imposing the constraint. To reduce the number of constraints, we substitute rjtkfr om Equation (f) into Equations(a) and (b) to obtain the following formulation (Ku and Karimi, 1988).

Minimize: CNM

Subject to: Equations (c), (d), (e) and

C„, r 0 and Xi„ binary Because the preceding formulation involves binary (XiJ) as well as continuous variables(Ci„) and has no nonlinear functions, it is a mixed-integer linear programming(MILP) problem and can be solved using the GAMS MIP solver. Solving for the optimal sequence using Table E16.2, we obtain XI„ = X2,4 = X3„ = X„, = 1. This means that pl is in the first position in the optimal production sequence, p2 in the fourth, p3 in the second, and p4 in the third. In other words, the optimal sequence is in the order pl-p3-p4-p2. In contrast to the XiJ, we must be careful in interpreting the Ci,f,ro m the GAMS output, because C„, really means the time at which the jth product in the sequence (and not product pi) leaves unit k. Therefore C2„ = 23.3 means that the second product (i.e., p3) leaves unit 3 at 23.3 h. Interpreting the others in this way, the schedule corresponding to this production sequence is conveniently displayed in form of a

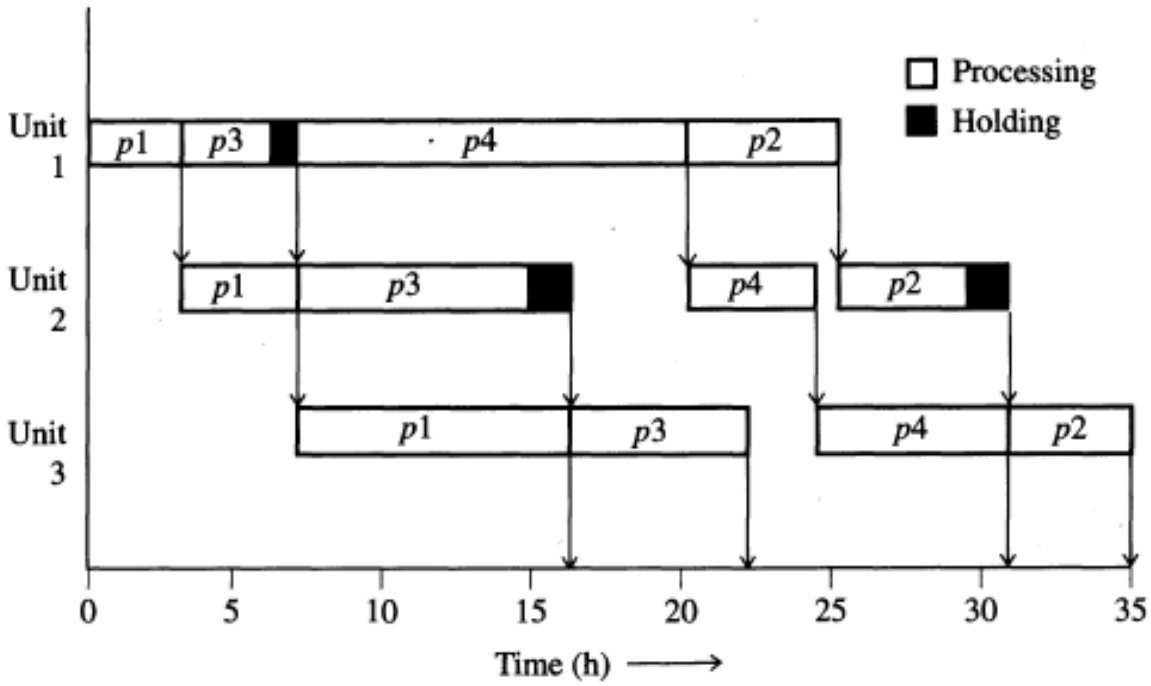


Figure 6.1: Gantt chart for the optimal multiproduct plant schedule

Gantt chart in Figure E16.2b, which shows the status of the units at different times. For instance, unit 1 is processing pl during [0, 3.51 h. When pl leaves unit 1 at t = 3.5 h, it starts processing p3. It processes p3 during [3.5,7] h. But as seen from the chart, it is unable to dischargep3 to unit 2, because unit 2 is still processing pl. So unit 1 holds p3 during [7,7.8] h. When unit 2 discharges p3

to unit 3 at 16.5 h, unit 1 is still processingp4, therefore unit 2 remains idle during [16.5, 19.81 h. It is common in batch plants to have units blocked due to busy downstream units or units waiting for upstream units to finish. This happens because the processing times vary from unit to unit and from product to product, reducing the time utilization of units in a batch plant. The finished batches of pl, p3, p4, and p2 are completed at times 16.5 h, 23.3 h, 3 1.3 h, and 34.8 h. The minimum makespan in 34.8 h.

This problem can also be solved by a search method . Because the order of products cannot be changed once they start through the sequence of units, we need only determine the order in which the products are processed. Let be a permutation or sequence in which to process the jobs, where p(j) is the index of the product in position j of the sequence. To evaluate the makespan of a sequence, we proceed as in Equations (a)-(c) of the mixed-integer programming version of the problem. Let Ci, be the completion time of product p(j) on unit k. If product p(j) does not have to wat for product p(j - 1) to finish its processing on unit k, then If it does have to wait, then Hence Cj,k is the larger of these

two values: This equation is solved first for Cl,kf or k = 1, . . . ,M , then for C,& for k = 1,2, . . . , M, and so on. The objective function is simply the completion time of the last job: In a four-product problem, there are only 4! = 24 possible sequences, so you can easily write a simple FORTRAN or C program to evaluate the makespan for an arbitrary sequence, and then call it 24 times and choose the sequence with the smallest makespan. For larger values of N, one can apply the tabu search algorithm . Other search procedures (e.g . , evolutionary algorithms or simulated annealing), can also be developed for this problem. Of course, these algorithms do not guarantee that an optimal solution will be found. On the other hand, the time required to solve the mixed-integer programming h u l a t i o n grows rapidly with N, so that approach eventually becomes impractical. This illustrates that you may be able to develop a simple but effective search method yourself, and eliminate the need for MILP optimization software.

The classical solution to a scheduling problem assumes that the required information is known at the time the schedule is generated and that this a priori scheduling remains fixed for a planning period and is implemented on the plant equipment. Although this methodology does not compensate for the many external disturbances and internal disruptions that occur in a real plant, it is still the strategy most commonly found in industrial practice. Demand fluctuations, process devia tions, and equipment failure all result in schedule infeasibilities that become apparent during the implementation of the schedule. To remedy this situation, frequent rescheduling becomes necessary.

In the rolling horizon rescheduling approach (Baker, 1993), a multiperiod solution is obtained, but only the first period is implemented. After one period has elapsed, we observe the existing inventories, create new demand forecasts, and solve a new multiperiod problem. This procedure tries to compensate for the fixed nature of the planning model. However, as has been pointed out by Pekny and Reklaitis (1998), schedules generated in this fashion generally result in frequent resequencing and reassignment of equipment and resources, which may induce further changes in successive schedules rather than smoothing out the production output.

An alternative approach uses a master schedule for planning followed by a reactive scheduling strategy to accommodate changes by readjusting the master schedule in a least cost or least change way. The terms able to promise or available to promise (ATP) indicate whether a given customer, product, volume, date, or time request can be met for a potential order. ATP requests might be filled from inventory, unallocated planned production, or spare capacity (assuming additional production). When the production scheduler is content with the current plan, made up of firm orders and forecast orders, the forecast orders are removed but the planned production is left intact. This produces inventory profiles in the model that represent ATP from inventory and from unallocated

planned production (Baker, 1993; Smith, 1998).

An important simulation tool used in solving production planning and scheduling problems is the discrete event dynamic system (DEDS), which gives a detailed picture of the material flows through the production process. Software for simulating such systems are called discrete event simulators. In many cases, rules or expert systems are used to incorporate the experience of scheduling and planning personnel in lieu of a purely optimization-based approach to scheduling (Bryant, 1993). Expert systems are valuable to assess the effects of changes in suppliers, to locate bottlenecks in the system, and to ascertain when and where to introduce new orders. These expert systems are used in reactive scheduling when fast decisions need to be made, and there is no time to generate another optimized production schedule.

6.5 Plantwide Management and Optimization

At the plantwide management and optimization level, engineers strive for enhancements in the operation of the equipment once it is installed in order to realize the most production, the greatest profit, the minimum cost, the least energy usage, and so on. In plant operations, benefits arise from improved plant performance, such as improved yields of valuable products (or reduced yields of contaminants), better product quality, reduced energy consumption, higher processing rates, and longer times between shut downs. Optimization can also lead to reduced maintenance costs, less equipment wear, and better staff utilization. Optimization can take place plantwide or in combinations of units.

The application of real-time optimization (RTO) in chemical plants has been carried out since the 1960s. Originally a large mainframe computer was used to optimize process setpoints, which were then sent to analog controllers for implementation.

In the 1970s this approach, called supervisory control, was incorporated into computer control systems with a distributed microprocessor architecture called a distributed control system, or DCS (Seborg et al., 1989). In the DCS both supervisory control and regulatory (feedback) control were implemented using digital computers. Because computer power has increased by a factor of lo6 over the past 30 years, it is now feasible to solve meaningful optimization problems using advanced tools such as linear or nonlinear programming in real time, meaning faster than the time between setpoint changes.

In RTO (level 3), the setpoints for the process operating conditions are optimized daily, hourly, or even every minute, depending on the time scale of the process and the economic incentives to make changes. Optimization of plant operations determines the setpoints for each unit at the temperatures, pressures, and flow rates that are the best in some sense. For example, the selection of the percentage

of excess air in a process heater is quite critical and involves a balance on the fuel-air ratio to ensure complete combustion and at the same time maximize use of the heating potential of the fuel. Examples of periodic optimization in a plant are minimizing steam consumption or cooling water consumption, optimizing the reflux ratio in a distillation column, blending of refinery products to achieve desirable physical properties, or economically allocating raw materials. Many plant maintenance systems have links to plant databases to enable them to track the operating status of the production equipment and to schedule calibration and maintenance. Real-time data from the plant also may be collected by management information systems for various business functions.

The objective function in an economic model in RTQ involves the costs of raw materials, values of products, and costs of production as functions of operating conditions, projected sales or interdepartmental transfer prices, and so on. Both the operating and economic models typically include constraints on

(a) Operating Conditions: Temperatures and pressures must be within certain limits.

(b) Feed and Production Rates: A feed pump has a fixed capacity; sales are limited by market projections.

(c) Storage and Warehousing Capacities: Storage tanks cannot be overfilled 'during periods of low demand.

(d) Product Impurities: A product may contain no more than the maximum amount of some contaminant or impurity.

In addition, safety or environmental constraints might be added, such as a temperature limit or an upper limit on a toxic species. Several steps are necessary for implementation of RTO, including determining the plant steady-state operating conditions, gathering and validating data, updating of model parameters (if necessary) to match current operations, calculating the new (optimized) setpoints, and implementing these setpoints. An RTO system completes all data transfer, optimization calculations, and setpoint implementations before unit conditions change and require a new optimum to be calculated.

Some of the RTO problems characteristic of level 3 are

1. Reflux ratio in distillation .
2. Olefin manufacture .
3. Ammonia synthesis .
4. Hydrocarbon refrigeration .

The last example is particularly noteworthy because it represents the current state of the art in utilizing fundamental process models in RTO. Another activity in RTO is determining the values of certain empirical parameters in process models from the process data after ensuring that the process is at steady state. Measured variables including

flow rates, temperatures, compositions, and pressures can be used to estimate model parameters such as heat transfer coefficients, reaction rate coefficients, catalyst activity, and heat exchanger fouling factors.

Usually only a few such parameters are estimated online, and then optimization is carried out using the updated parameters in the model. Marlin and Hrymak (1997) and Forbes et al. (1994) recommend that the updated parameters be observable, represent actual changes in the plant, and significantly influence the location of the optimum; also the optimum of the model should be coincident with that of the true process. One factor in modeling that requires close attention is the accurate representation of the process constraints, because the optimum operating conditions usually lie at the intersection of several constraints. When RTO is combined with model predictive regulatory control (see Section 16.4), then correct (optimal) moves of the manipulated variables can be determined using models with accurate constraints.

Marlin and Hrymak (1997) reviewed a number of industrial applications of RTO, mostly in the petrochemical area. They reported that in practice a maximum change in plant operating variables is allowable with each RTO step. If the computed optimum falls outside these limits, you must implement any changes over several steps, each one using an RTO cycle. Typically, more manipulated variables than controlled variables exist, so some degrees of freedom exist to carry out both economic optimization as well as establish priorities in adjusting manipulated variables while simultaneously carrying out feedback control.

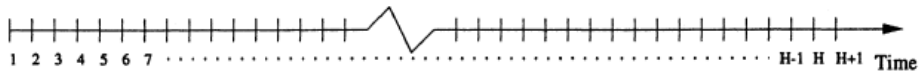
6.6 Resent trends in scheduling

Batch process industries in general are consisting of several products produced in more than one plant in the same premises. Specialty chemicals and pharmaceutical products are typically produced in batch processes. The production demand is not fixed; it will change as per the market demand. In these industries the production is decided by the market demand. These industries produce two or more products in one plant in same equipment. Batch process has reduced inventories and/ or shortened response time compare to continuous processes 15.

2.2 Scheduling of batch processes and representation of the process (RTN, STN):

Scheduling is a decision-making process that concerns the allocation of limited resources to competing tasks over time with the goal of optimizing one or more objectives. The general scheduling problem can be posed as follows:

- Production facility data; e.g., processing unit and storage vessel capacities, utility availability, unit connectivity.



Discrete-time representation

- Detailed production recipes; e.g., stoichiometric coefficients, processing times, processing rates, utility requirements.
- Production costs; e.g., raw materials, utilities, cleaning, etc.
- Production targets or orders with due dates.

The goal of scheduling is to determine

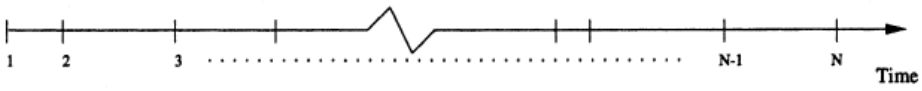
- o The allocation of resources to processing tasks.
- o The sequencing and timing of tasks on processing units.

The objectives functions include the minimization of make span, lateness and earliness, as well as the minimization of total cost. Scheduling formulations can be broadly classified in to discrete-time models and continuous time models 16.

Early attempts in modeling the process scheduling problems relied on the discrete-time approach, in which the time horizon is divided in to a number of time intervals of uniform durations and events such as the beginning and ending of a task are associated with the boundaries of these time intervals. To achieve a suitable approximation of the original problem, it is usually needed to use a time interval that is sufficiently small, for example, the greatest common factor (GCF) of the processing times. This usually leads to very large combinatorial problems of intractable size, especially for real-world problems, and hence limits its applications. The basic concept of the discrete-time approach is illustrated in Fig (2.1).

Continuous-time models events are potentially allowed to take place at any point in the continuous domain of time. Modeling of this flexibility is accomplished by introducing the concepts of variable event times, which can be defined globally or for each unit. Variables are required to determine the timings of events. The basic idea of the continuous-time approach is also illustrated in Fig (2.2). Because of the possibility of eliminating a major fraction of the inactive event-time interval assignments with the continuous-time approach, the resulting mathematical programming problems are usually of much smaller sizes and require less computational efforts for their solution. However, due to the variable nature of the timings of the events, it becomes more challenging to model the scheduling process and the continuous-time approach may lead to mathematical models with more complicated structures compared to their discrete-time counterparts.

Network represented processes are involve in most scheduling problems. When the production recipes become more complex and/or different products have low recipe sim-



Continuous-time representation

ilarities, processing networks are used to represent the production sequences. This corresponds to a more general case in which batches can merge and/or split and material balances are required to be taken in to account explicitly. The state-task network and resource-task network are generally used in scheduling problems 17.

Model -based batch scheduling methodologies can be grouped into two types: monolithic and sequential approaches Monolithic approaches are those which simultaneously determine the set of tasks to be scheduled, the allocation of manufacturing resources to tasks and the sequence of tasks at any equipment unit. Monolithic scheduling models are quite general; they are more oriented towards the treatment of arbitrary network processes involving complex product recipes. They are based on either the state-task network (STN) or the resource-task network (RTN) concepts to describe production recipes 18.

6.6.1 State-Task Network (STN):

The established representation of batch processes is in terms of "recipe networks" 4. These are similar to the flow sheet representation of continuous plants but are intended to describe the process itself rather than a specific plant. Each node on a recipe network corresponds a task, with directed arcs between nodes representing task precedence. Although recipe networks are certainly adequate for serial processing structures, they often involve ambiguities when applied to more complex ones. Consider, for instance, the network of five tasks shown in Fig (2.3).

It is not clear from this representation whether task 1 produces two different products, forming the inputs of tasks 2 and 3 respectively, or whether it has only one type of product which is then shared between 2 and 3. Similarly, it is also impossible to determine from Fig (2.3) whether task 4 requires two different types of feed stocks, respectively produced by tasks 2 and 5, or whether it only needs one type of feedstock which can be produced by either 2 or 5. Both interpretations are equally plausible. The former could be the case if, say, task 4 is a catalytic reaction requiring a main feedstock produced by task 2, and a catalyst which is then recovered from the reaction products by the separation task 5. The latter case could arise if task 4 were an ordinary reaction task with a single feedstock produced by 2, with task 5 separating the product from the unreacted material which is

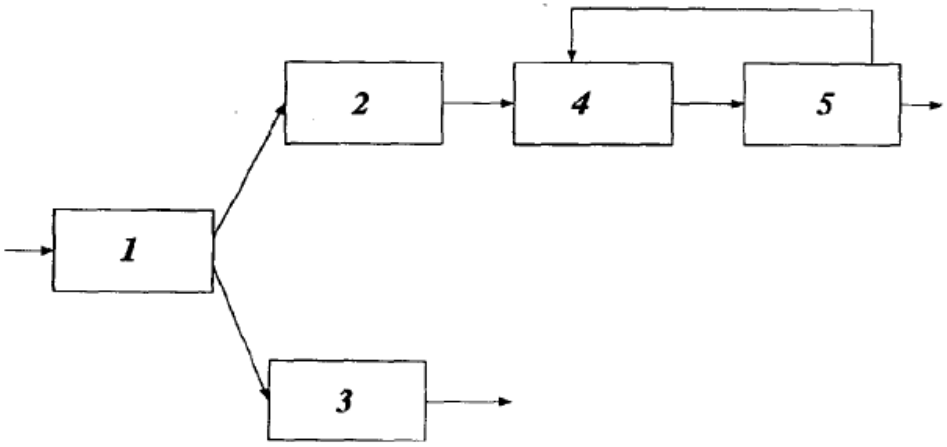


Figure 6.2: Recipe networks

then recycled to 4.

The distinctive characteristic of the STN is that it has two types of nodes; namely, the state nodes, representing the feeds, intermediate and final products and the task nodes, representing the processing operations which transform material from one or more input states to one or more output states. State and task nodes are denoted by circles and rectangles, respectively.

State-task networks are free from the ambiguities associated with recipe networks. Figure (2.4) shows two different STNs, both of which correspond to the recipe network of Fig (2.3). In the process represented by the STN of Fig (2.4a), task 1 has only one product which is then shared by tasks 2 and 3. Also, task 4 only requires one feedstock, which is produced by both 2 and 5. On the other hand, in the process shown in Fig (2.4b), task 1 has two different products forming the inputs to tasks 2 and 3, respectively. Furthermore, task 4 also has two different feedstocks, respectively produced by tasks 2 and 5.

Bhattacharya et.al. (2009) proposes a mathematical model based on State Task Network representation for generating optimal schedule for a sequence of n continuous processing units responsible for processing m products. Each product line has fixed capacity storage tanks before and after every unit. Each unit can process only one product line at any point of time. However, the inputs for all the product lines arrive simultaneously at the input side of the first unit, thus giving rise to the possibility of spillage. There can be multiple intermediate upliftments of the finished products. An optimal schedule of the units attempts to balance among spillage, changeover and upliftment failures. They develop an MINLP model for the problem 19.

Ierapetritou et.al. (2004) proposed a new cyclic scheduling approach is based on the state-task network (STN) representation of the plant and a continuous-time formulation.

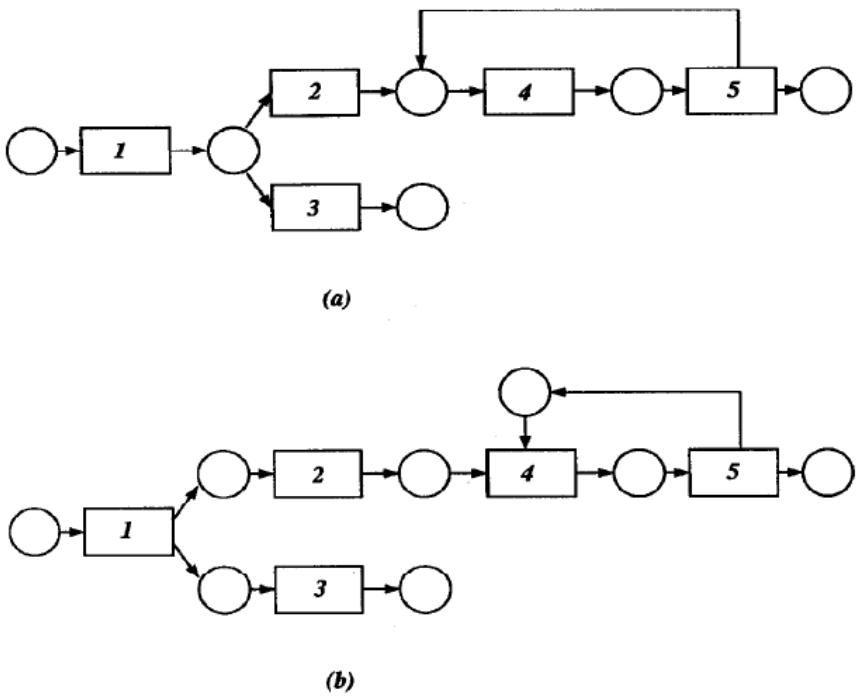


Figure 6.3: STN representation

Assuming that product demands and prices are not fluctuating along the time horizon under consideration, the proposed formulation determines the optimal cycle length as well as the timing and sequencing of tasks within a cycle. This formulation corresponds to a non-convex mixed integer nonlinear programming (MINLP) problem, for which local and global optimization algorithms are used and the results are illustrated for various case studies 7.

6.6.2 Resource – Task Network (RTN):

RTN-based mathematical formulations can either be discrete-time or continuous-time formulations. The original is a discrete-time formulation, where the time horizon of interest is discretized into a fixed number (T) of uniform time intervals. The RTN regards all processes as bipartite graphs comprising two types of nodes: Resources and Tasks. A task is an operation that transforms a certain set of resources into another set. The concept of resource is entirely general and includes all entities that are involved in the process steps, such as materials (raw-materials, intermediates and products), processing and storage equipment (tanks, reactors, etc.), utilities (operators, steam, etc.) as well as equipment conditions (clean, dirty).

The simple motivating example involving a multipurpose batch plant that requires one raw material and produces two intermediates and one final product. The RTN for

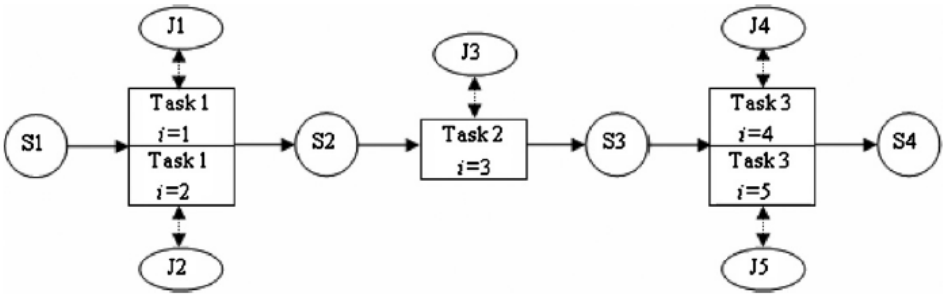


Figure 6.4: RTN representation

this example is presented by Shaik et.al. (2008) shown in Fig (2.5). The raw material is processed in three sequential tasks, where the first task is suitable in two units (J1 and J2), the second task is suitable in one unit (J3), and the third task is suitable in two units (J4 and J5). The equipment resources (units) are shown using double arrow connectors, indicating that the task consumes this resource at the beginning of the event and produces the same resource at the end of the event point. A task which can be performed in different units is considered as multiple, separate tasks, thus leading to five separate tasks (i=1, . . ., 5), each suitable in one unit 3.

Shaik et.al. (2008) propose a new model to investigate the RTN representation for unit-specific event-based models. For handling dedicated finite storage, a novel formulation is proposed without the need for considering storage as a separate task. The performance of the proposed model is evaluated along with several other continuous-time models from the literature based on the STN and RTN process representations 3.

Grossmann et.al. (2009) consider the solution methods for mixed-integer linear fractional programming (MILFP) models, which arise in cyclic process scheduling problems. Dinkelbach's algorithm is introduced as an efficient method for solving large-scale MILFP problems for which its optimality and convergence properties are established. Extensive computational examples are presented to compare Dinkelbach's algorithm with various MINLP methods. To illustrate the applications of this algorithm, we consider industrial cyclic scheduling problems for a reaction–separation network and a tissue paper mill with byproduct recycling. These problems are formulated as MILFP models based on the continuous time Resource-Task Network (RTN) 21.

6.6.3 Optimum batch schedules and problem formulations (MILP),MINLP B&B:

Grossmann et.al. (2006) presents a multiple time grid continuous time MILP model for the short-term scheduling of single stage, multi product batch plants. It can handle both release and due dates and the objective can be either the minimization of total cost or total earliness. This formulation is compared to other mixed-integer linear programming approaches, to a constraint programming model, and to a hybrid mixed-integer linear/constraint programming algorithm. The results show that the proposed formulation is significantly more efficient than the MILP and CP models and comparable to the hybrid model 22.

Mixed Integer Linear Programming (MILP) model is used for the solution of N-dimensional allocation problems. Westerlund et.al. (2007), use Mixed Integer Linear Programming (MILP) model and solve the one-dimensional scheduling problems, two-dimensional cutting problems, as well as plant layout problems and three-dimensional packing problems. Additionally, some problems in four dimensions are solved using the considered model 23.

Magatao et.al. (2004), present the problem of developing an optimisation structure to aid the operational decision-making of scheduling activities in a real-world pipeline scenario. The pipeline connects an inland refinery to a harbour, conveying different types of oil derivatives. The optimisation structure is developed based on mixed integer linear programming (MILP) with uniform time discretisation, but the MILP well-known computational burden is avoided by the proposed decomposition strategy, which relies on an auxiliary routine to determine temporal constraints, two MILP models, and a database 24.

A new mixed-integer programming (MIP) formulation is presented for the production planning of single-stage multi-product processes. The problem is formulated as a multi-item capacitated lot-sizing problem in which (a) multiple items can be produced in each planning period, (b) sequence-independent set-ups can carry over from previous periods, (c) set-ups can cross over planning period boundaries, and (d) set-ups can be longer than one period. The formulation is extended to model time periods of non-uniform length, idle time, parallel units, families of products, backlogged demand, and lost sales 25.

Bedenik et.al. (2004) describes an integrated strategy for a hierarchical multilevel mixed-integer nonlinear programming (MINLP) synthesis of overall process schemes using a combined synthesis/analysis approach. The synthesis is carried out by multilevel-hierarchical MINLP optimization of the flexible superstructure, whilst the analysis is performed in the economic attainable region (EAR). The role of the MINLP synthesis

step is to obtain a feasible and optimal solution of the multi-D process problem, and the role of the subsequent EAR analysis step is to verify the MINLP solution and in the feedback loop to propose any profitable superstructure modifications for the next MINLP. The main objective of the integrated synthesis is to exploit the interactions between the reactor network, separator network and the remaining part of the heat/energy integrated process scheme 26.

Grossmann et.al. (2009) consider the solution methods for mixed-integer linear fractional programming (MILFP) models, which arise in cyclic process scheduling problems. They first discuss convexity properties of MILFP problems, and then investigate the capability of solving MILFP problems with MINLP methods. Dinkelbach's algorithm is introduced as an efficient method for solving large-scale MILFP problems for which its optimality and convergence properties are established. Extensive computational examples are presented to compare Dinkelbach's algorithm with various MINLP methods 21.

Cyclic scheduling is commonly utilized to address short-term scheduling problems for multi product batch plants under the assumption of relatively stable operations and product demands. It requires the determination of optimal cyclic schedule, thus greatly reducing the size of the overall scheduling problems with large time horizon. The proposed formulation determines the optimal cycle length as well as the timing and sequencing of tasks within a cycle. This formulation corresponds to a non-convex mixed integer nonlinear programming (MINLP) problem, for which local and global optimization algorithms 7. Floudas et.al. (1998a) presents a novel mathematical formulation for the short-term scheduling of batch plants. The proposed formulation is based on a continuous time representation and results in a mixed integer linear programming (MILP) problem 27.

Reklaitis et.al. (1999) proposed a MINLP formulation using the state-task network (STN) representation. They proposed a Bayesian heuristic approach to solve the resulting non convex model 4. Grossmann et.al. (1991) presented a reformulation of the multi-period based MILP model based on lot-sizing. The main strength of this formulation is that it provides a tighter LP relaxation leading to better solutions than the earlier MILP formulation 28.

Kondili et.al. (1993) studied the problem of planning a multi product energy-intensive continuous cement milling plant. They utilized a hybrid discrete-continuous time formulation considering discrete time periods and time slots of varying duration. The resulting mixed integer linear programming (MILP) problem was solved with conventional MILP solvers based on branch and bound (B&B) principles. Towards developing efficient mathematical models to address the short-term scheduling problem, attention has been given to continuous-time representations 29. Floudas et.al. (1998a, 1998b) developed

a continuous-time formulation for the short-term scheduling problem that requires less number of variables and constraints 27, 30.

6.6.4 Multi-product batch plants:

Multistage, multi-product batch plants with parallel units in one or more stages abound in the batch chemical industry. In such plants, scheduling of operations is an essential, critical, and routine activity to improve equipment utilization, enhance on-time customer delivery, reduce setups and waste, and reduce inventory costs. The most research on batch process scheduling has focused on serial multi product batch plants or single-stage non continuous plants, scheduling of multistage, multi product batch plants has received limited attention in the literature in spite of the industrial significance of these plants 7. Multi product batch plants, in which every product follows the same sequence through all the process steps, or as multi-purpose batch plants, in which each product follows its own distinct processing sequence by using the available equipment in a product-specific layout 1.

Kondili et.al. (1993) studied the problem of planning a multi product energy-intensive continuous cement milling plant. They utilized a hybrid discrete-continuous time formulation considering discrete time periods and time slots of varying duration [29].

The cyclic scheduling of cleaning and production operations in multi product multistage plants with performance decay. A mixed-integer nonlinear programming (MINLP) model based on continuous time representation is proposed that can simultaneously optimize the production and cleaning scheduling. The resulting mathematical model has a linear objective function to be maximized over a convex solution space thus allowing globally optimal solutions to be obtained with an outer approximation algorithm 7. Majozi et.al. (2009) presents a methodology for wastewater minimisation in multipurpose batch plants characterized by multiple contaminant streams. The methodology is based on an existing scheduling framework which then makes it possible to generate the required schedule to realize the absolute minimum wastewater generation for a problem. The methodology involves a two step solution procedure. In the first step the resulting MINLP problem is linearized and solved to provide a starting point for the exact MINLP problems 31.

.

6.6.5 Waste water minimization (Equalization tank super structure):

Wastewater minimization in batch plants is gaining importance as the need for industry to produce the least amount of effluent becomes greater. This is due to increasingly stringent environmental legislation and the general reduction in the amount of available freshwater sources. Wastewater minimization in batch plants has not been given sufficient attention in the past. Smith et.al. (1995) proposed one of the first methodologies for wastewater minimization in batch processes 32. Smith et.al. (1994) proposed methodology was an extension of their graphical water pinch method. This method dealt with single contaminant wastewater minimization, where it was assumed that the optimal process schedule was known a priori. The method employed a number of water storage vessels to overcome the discontinuous nature of the operation 33.

Mazozi et.al. (2005a) proposed a mathematical methodology where the corresponding plant schedule is determined that achieves the wastewater target. In doing this the true minimum wastewater target can be identified 34. This methodology was latter extended to include multiple contaminants with multiple storage vessels by Majozi et.al. (2008) 35. The methodology presented by Majozi et.al. (2009) explores the usage idle processing units as inherent storage for wastewater. The methodology is derived for the single contaminant case and can deal with two types of problems. In the first type of problem the minimum wastewater target and corresponding minimum wastewater storage vessel is determined considering inherent storage. The second problem is centred on the determination of the minimum wastewater target where the size of the central storage vessel is fixed and inherent storage is available. In the second type of problem the size of the central storage limits wastewater reuse 31.

Most of the studies published in literature have dealt with the issue of minimizing waste using processes separately from the design of effluent treatment systems. A conceptual approach has been used to minimize the wastewater generation in process industries 33. Graphical techniques have been presented to set targets for the minimum flow rate in a distributed effluent system and to design such systems 36, 37.

Shoaib et.al. (2008) addresses the problem of synthesising cost-effective batch water networks where a number of process sources along with fresh water are mixed, stored, and assigned to process sinks. In order to address the complexity of the problem, a three-stage hierarchical approach is proposed. In the first stage, global targets are identified by formulating and solving a linear transportation problem for minimum water usage, maximum water recycle, and minimum wastewater discharge. These targets are determined a priori and without commitment to the network configuration 38.

Manan et al., (2004) has proved its validity in identifying correct water network targets that include the minimum fresh water and wastewater flow rate targets, the global pinch location and water allocation targets. A key assumption in that work is that water sources of different impurity concentrations are not allowed to mix in the same water storage tank. This is a limitation that may lead to an excessive number of storage tanks. A water stream chart which can identify the possibility for direct water reuse was built. This model included tank capacity, stream concentration upper bounds and time constraints 39.

Chang et.al. (2006) proposed a general mathematical programming model is developed to design the optimal buffer system for equalizing the flow-rates and contaminant concentrations of its outputs. The demands for heating/cooling utilities in a batch plant arise intermittently and their quantities vary drastically with time, the generation rates of the resulting spent waters must also be time dependent. A buffer tank can thus be used at the entrance of each utility system to maintain a steady throughput. The capital cost of a wastewater treatment operation is usually proportional to its capacity. Thus, for economic reasons, flow equalization is needed to reduce the maximum flow-rate of wastewater entering the treatment system 10.

A buffer system may also be installed to equalize the wastewater flow-rates and pollutant concentrations simultaneously. The inputs of this equalization system can be the spent utility waters or wastewaters generated from various batch operations, and the outputs can be considered to be the feeds to different utility-producing equipments, wastewater-treatment units and/or discharge points 10.

Li et.al. (2002a,b) adopted both a conceptual design approach and also a mathematical programming model to eliminate the possibility of producing an unnecessarily large combined water flow at any instance by using a buffer tank and by rescheduling the batch recipe 13, 14. Later, Li et.al. (2003) used a two-tank configuration to remove peaks in the profile of total wastewater flow-rate and also in that of one pollutant concentration [40.

A proper design of the water equalization system includes at least the following specifications:

(1) The needed size of each buffer tank,

(2) The network configuration, and

(3) The time profiles of the flow-rate and pollutant concentrations of the water stream in every branch of the equalization network.

Super structure of Equalization system presented by Chang et.al. (2006):

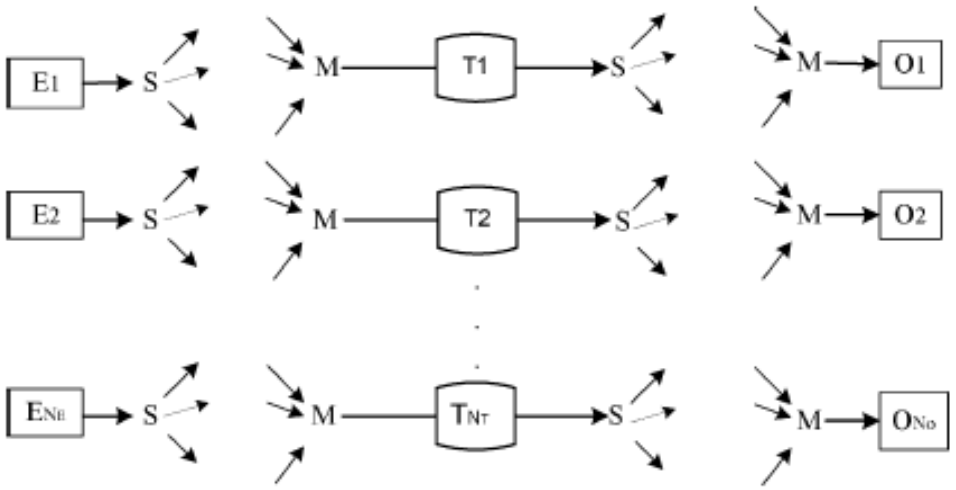


Figure 6.5: Super structure of equilisation tanks

6.6.6 Selection of suitable equalization tanks for controlled flow:

The effluent generated from batch plant is treated in a continuous operated effluent treatment plant. Discharge of the effluent generated from each one of these plant is discontinuous in nature. Quality and quantity of the effluent generated depend on the products produced and their quantities. In such case selection of suitable equalization tanks plays major role. Select a equalization tank in such a way that the overflow will not occur and the contaminant concentration will not to high in the discharge of this equalization tank under the following conditions.

- If the production quantities are drastically increase
- If the batch failure will occur.
- If the sudden shout down has taken place

The equalization tank design in such a way that the contaminant concentrations will high then the concentration of effluent will be equalize. In some cases two equalization tanks are needed, one is lower concentration and another is higher concentration. So that the effluent discharge from the equalization tank has equalized concentration and suitable flow rate are going to effluent treatment plant and decrease the shock load of ETP.

Chapter 7

Dynamic Programming

Interest in dynamic simulation and optimization of chemical processes has increased significantly in process industries over the past two decades. This is because process industries are driven by strongly competitive markets and have to face ever-tighter performance specifications and regulatory limits. The transient behavior of chemical processes is typically modeled by ordinary differential equations (ODEs), differential/algebraic equations (DAEs), or partial differential/algebraic equations (PDAEs). These equations describe mass, energy, and momentum balances and thus ensure physical and thermodynamic consistency. Due to the now widespread industrial use of dynamic process modeling, dynamic models are increasingly developed and applied. It is then natural to ask what other applications might exploit the resources invested in the development of such models. Dynamic optimization is a natural extension of these dynamic simulation tools because it automates many of the decisions required for engineering studies. It consists of determining values for input/control profiles, initial conditions, and/or boundary conditions of a dynamic system that optimize its performance over some period of time. Examples of dynamic optimization tasks include: determination of optimal control profiles for batch processes; determination of optimal changeover policies in continuous processes, optimizing feeding strategies of various substrates in semi batch fermenters to maximize the productivity, and fitting of chemical reaction kinetic parameters to data.

A general formulation of the dynamic optimization problem can be stated as,

$$\min_{u(t),t_f} J = G(x(t_f), y(t_f), u(t_f) + \int L(x(t)), y(t), u(t), t), t \in [t_o, t_f] \tag{7.1}$$

$$h \quad : \quad s.t.x = f(x,u) \tag{7.2}$$

$$y(t) \quad = \quad l(x,u) \tag{7.3}$$

$$x(to) : x_o \tag{7.4}$$

$$g \quad : \quad y_{\min} \leq y(t) \leq y_{\max} \tag{7.5}$$

$$u_{\,min} \quad \leq \quad u(t) \leq u_{\max} \tag{7.6}$$

where t0 and tf denote the initial and final transition times and vectors x(t), y(t) and u(t) represent the state, output and input trajectories. Vector functions h and g are used to denote all equality and inequality constraints, respectively. Equations 1b,c represent the process model, whereas equation 1d represents the process and safety constraints. The solution of the above problem yields dynamic profiles of manipulated variables u(t) as well as the grade changeover time, tf – t0. Equation 1 may be solved with a standard NLP solver through use of CVP, where manipulated variables are parameterized and approximated by a series of trial functions . Thus, for the ith manipulated variable,

$$u_i(t) = \sum_{j=1}^{na} a_{ij}\gamma_{ij}(t - t_{ij}) \tag{7.7}$$

where tij is the jth switching time of the ith manipulated variable, na is the number of switching intervals, and aij represents the amplitude of the ith manipulated variable at the switching time tij.

necessary steps that integrate CVP with the model equations and the NLP solver are summarized below,

Step 1. Discretize manipulated variables by selecting an appropriate trial function (see equation 2)

Step 2. Integrate the process model (Equation 1b,c) and the ODE sensitivities (Equations 3) if using a gradient-based solver and the manipulated variables as inputs

Step 3. Compute the objective function (Equation 1a) and gradients if necessary

Step 4. Provide this information to the NLP solver and iterate starting with step 2 until an optimal solution is found

Step 5. Construct the optimal recipes using the optimal amplitude and switching intervals obtained by the NLP solver.

An alternative strategy eliminates Step 2 by discretizing the continuous process model by simultaneously treating both, the state variables and the parameterized manipulated variables as decision variables.

In this workshop, we will focus on problem formulation, solution approach, and demonstrating its application to a polymerization reactor for optimal product grade transition. We shall use MATLAB tool for solving a dynamic optimization problem solution.

Chapter 8

Global Optimisation Techniques

8.1 Introduction

Tradictional optimisation techniques have the limitation of getting traped in the local optimum. To over come this problem some optimisation techniques based on the natural phenomena are proposed. Some stocastic optimisation techniques which are popular are presented in the following sections

8.2 Simulated Annealing

8.2.1 Introduction

This optimization technique resembles the cooling process of molten metals through annealing. The atoms in molten metal can move freely with respect to each other at high temperatures. But the movement of the atoms gets restricted as the temperature is reduced. The atoms start to get ordered and finally form crystals having the minimum possible energy. However, the formation of crystals mostly depends on the cooling rate. If the temperature is reduced at a very fast rate, the crystalline state may not be achieved at all, instead, the system may end up in a polycrystalline state, which may have a higher energy than the crystalline state. Therefore, in order to achieve the absolute minimum energy state, the temperature needs to be reduced at a slow rate. The process of slow cooling is known as annealing in metallurgical parlance.

The simulated annealing procedure simulates this process of slow cooling of molten metal to achieve the minimum value of a function in a minimization problem. The cooling phenomenon is simulated by controlling a temperature-like parameter introduced with the concept of the Boltzmann probability distribution. According to the Boltzmann probability distribution, a system in thermal equilibrium at a temperature T has its

energy distributed probabilistically according to P(E) = exp (-EAT), where k is the Boltzmann constant. This expression suggests that a system at high temperature has almost uniform probability of being in any energy state, but at low temperature it has a small probability of being in a high energy state.

Therefore, by controlling the temperature T and assuming that the search process follows the Boltzmann probability distribution, the convergence of an algorithm can be controlled. Metropolis et al. (1953) suggested a way to implement the Boltzmann probability distribution in simulated thermodynamic systems. This can also be used in the function minimization context. Let us say, at any instant the current point is x") and the function value at that point is E(t) =f(x(')). Using the Metropolis algorithm, we can say that the probability of the next point being at

x" + ') depends on the difference in the function values at these two points or on AE = E(f + 1) -E(f) and is calculated using the Boltzmann probability distribution: If c\E I 0, this probability is 1 and the point x(' + ') is always accepted. In the function minimization context, this makes sense because if the function value at x" + ') is better than that at x('), the point x(' + ') must be accepted. An interesting sjtuation results when AE > 0, which implies that the function value at x(' + ') is worse than that at x('). According to many traditional algorithms, the point x'" ') must not be chosen in this situation. But according to the Metropolis algorithm, there is some finite probability of selecting the point x('+ even though it is worse than the point x('). However, this probability is not the same in all situations. This probability depends on the relative magnitude of AE and T values. If the parameter T is large, this probability is more or less high for points with largely disparate function values. Thus, any point is almost acceptable for a large value of T. On the other hand, if the parameter T is small, the probability of accepting an arbitrary point is small. Thus, for small values of T, the points with only a small deviation in the function value are accepted.

Simulated annealing (Laarhoven & Aarts 1987) is a point-by-point method. The algorithm begins with an initial point and a high temperature T. A second point is created at random in the vicinity of the initial point and the difference in the function values (AE) at these two points is calculated. If the second point has a smaller function value, the point is accepted; otherwise the point is accepted with probability exp (-AE/T). This completes one iteration of the simulated annealing procedure. In the next generation, another point is created at random in the neighbourhood of the current point and the Metropolis algorithm is used to accept or reject the point. In order to simulate the thermal equilibrium at every temperature, a number of points (n) are usually tested at a particular temperature before reducing the temperature. The algorithm is terminated when a sufficiently small temperature is obtained or a small enough change in function

values is found.

Step 1 Choose an initial point X('), a termination criterion E. Set T sufficiently high, number of iterations to be performed at a particular temperature n, and set t = 0.

Step 2 Calculate a neighbouring point x"' ') = N(x(')). Usually, a random point in the neighbourhood is created.

Step 3 If AE = E(x(' + ") - E(x(')) c 0, set t = t + 1 ; else create a random number r in the range (0,l). If r 5 exp (-AE/T), set t = t + 1; else go to step 2.

Step 4 If Ix('+ - x(')l c E and T is small, terminate; else if (t mod n) = 0, then lower T according to a cooling schedule. Go to step 2; else go to step 2. The initial temperature T and the number of iterations (n) performed at a particular temperature are two important parameters which govern the successful working of the simulated annealing procedure. If a large initial T is chosen, it takes a number of iterations for convergence. On the other hand, if a small initial T is chosen, the search is not adequate to thoroughly investigate the search space before converging to the true optimum. A large value of n is recommended in order to achieve a quasi-equilibrium state at each temperature, but the computation time is more.

Unfortunately, there are no unique values of the initial temperature and n that work for every problem. However, an estimate of the initial temperature can be obtained by calculating the average of the function values at a number of random points in the search space. A suitable value of n can be chosen (usually between 20 and 100) depending on the available computing resources and the solution time. Nevertheless, the choice of the initial temperature and the subsequent cooling schedule still remain an art and usually require some trial-and-error efforts.

8.3 GA

8.3.1 Introduction

Genetic algorithms (GAS) are used primarily for optimization purposes. They belong to the goup of optimization methods known as non-traditional optimization methods.

Non-traditional optimization methods also include optimization techniques such as neural networks (simulated annealing, in particular). Simulated annealing is an optimization method that mimics the cooling of hot metals, which is a natural phenomenon. Through this mode of simulation, this method is utilized in optimization problems.

Just like simulated annealing, GAS try to imitate natural genetics and natural selection, which are natural phenomena. The main philosophy behind GA is 'survival of the fittest'. As a result of this, GAS are primarily used for maximization problems in

optimization.

GAS do not suffer from the basic setbacks of traditional optimization methods, such as getting stuck in local minima. This is because GAS work on the principle of natural genetics, which incorporates large amounts of randomness and does not allow stagnation.

8.3.2 Definition

Genetic algorithms are computerized search and optimization algorithms based on the mechanics of natural genetics and natural selection. Consider a maximization problem Maximize f (x) x;l) 5 xi 5 xi(u) i = 1, 2, ..., N

This is an unconstrained optimization problem. Our aim is to find the maximum of this function by using GAS. For the implementation of GAS, there are certain well-defined steps.

8.3.3 Coding

Coding is the method by which the variables xi are coded into string structures. This is necessary because we need to translate the range of the function in a way that is understood by a computer. Therefore, binary coding is used. This essentially means that a certain number of initial guesses are made within the range of the function and these are transformed into a binary format. It is evident that for each guess there might be some required accuracy. The length of the binary coding is generally chosen with respect to the required accuracy.

For example, if we use an eight-digit binary code. (0000 0000) would represent the minimum value possible and (1 11 1 11 11) would represent the maximum value possible.

A linear mapping rule is used for the purpose of coding:(/) x y - x!/) xi = xi + (decoded value) 2Il - 1

The decoded value (of si, the binary digit in the coding) is given by the rule / - I .I ˜ Decoded value = C2'si i=O

where

s; E (0, 1) (10.4)

For example, for a binary code (01 1 1), the decoded value will be (0111) = (112˜+ (112˜+ (1p2+ (012˜= 7

It is also evident that for a code of length 1, there are 2' combinations or codes possible. As already mentioned, the length of the code depends upon the required accuracy for that variable. So, it is imperative that there be a method to calculate the required string length for the given problem. So, in general, Eq. is used for this purpose.

So, adhering to the above-mentioned rules, it is possible to generate a umber of guesses or, in other words, an initial population of coded points that lie in the given range of the function. Then, we move on to the next step, that is, calculation of the fitness.

8.3.4 Fitness

As has already been mentioned, GAS work on the principle of 'survival of the fittest'. This in effect means that the 'good points' or the points that yield maximum values for the function are allowed to continue in the next generation, while the less profitable points are discarded from our calculations. GAS maximize a given function, so it is necessary to transform a minimization problem to a maximization problem before we can proceed with our computations.

Depending upon whether the initial objective function needs to be maximized or minimized, the fitness function is hence defined in the following ways: for a minimization problem It should be noted that this transformation does not alter the location of the minimum value. The fitness function value for a particular coded string is known as the string's fitness. This fitness value is used to decide whether a particular string carries on to the next generation or not.

GA operation begins with a population of random strings. These strings are selected from a given range of the function and represent the design or decision variables. To implement our optimization routine, three operations are carried out:

F(x) =Ax) for a maximization problem - 1 1 + f(x>

1. Reproduction
2. Crossover
3. Mutation

8.3.5 Operators in GA

Reproduction

The reproduction operator is also called the selection operator. This is because it is this operator that decides the strings to be selected for the next generation. This is the first operator to be applied in genetic algorithms. The end result of this operation is the formation of a 'mating pool', where the above average strings are copied in a probabilistic manner. The rule can be represented as a (fitness of string) Probability of selection into mating pool The probability of selection of the ith string into the mating pool is given by where Fi is the fitness of the ith string. Fj is the fitness of thejth string, and n is the population size.

The average fitness of all the strings is calculated by summing the fitness of individual strings and dividing by the population size, and is represented by the symbol F: - i=l F = - n

It is obvious that the string with the maximum fitness will have the most number of copies in the mating pool. This is implemented using 'roulette wheel selection'. The algorithm of this procedure is as follows. Roulette wheel selection

Step I Using Fi calculate pi.

Step 2 Calculate the cumulative probability Pi.

Step 3 Generate n random numbers (between 0 and 1).

Step 4 Copy the string that represents the chosen random number in the cumulative probability range into the mating pool. A string with higher fitness will have a larger range in the cumulative probability and so has more probability of getting copied into the mating pool.

At the end of this implementation, all the strings that are fit enough would have been copied into the mating pool and this marks the end of the reproduction operation.

Crossover

After the selection operator has been implemented, there is a need to introduce some amount of randomness into the population in order to avoid getting trapped in local searches. To achieve this, we perform the crossover operation. In the crossover operation, new strings are formed by exchange of information among strings of the mating pool. For example,

oo/ooo - 00˜111

Crossover point

l l j l l l

Parents Children

Strings are chosen at random and a random crossover point is decided, and crossover is performed in the method shown above. It is evident that, using this method, better strings or worse strings may be formed. If worse strings are formed, then they will not survive for long, since reproduction will eliminate them. But what if majority of the new strings formed are worse? This undermines the purpose of reproduction. To avoid this situation, we do not select all the strings in a population for crossover. We introduce a crossover probability (p,). Therefore, (loop,)% of the strings are used in crossover. (1 - p,)% of the strings is not used in crossover.

Through this we have ensured that some of the good strings from the mating pool remain unchanged. The procedure can be summarized as follows: Step I Select (loop,)%

of the strings out of the mating pool at random. Step 2 Select pairs of strings at random (generate random numbers that map the string numbers and select accordingly).

Step 3 Decide a crossover point in each pair of strings (again, this is done by a random number generation over the length of the string and the appropriate position is decided according to the value of the random number).

Step 4 Perform the crossover on the pairs of strings by exchanging the appropriate bits.

Mutation

Mutation involves making changes in the population members directly, that is, by flipping randomly selected bits in certain strings. The aim of mutation is to change the population members by a small amount to promote local searches when the optimum is nearby. Mutation is performed by deciding a mutation probability pm and selecting strings, on which mutation is to be performed, at random. The procedure can be summarized as follows:

Step 1 Calculate the approximate number of mutations to be performed by Approx no. of mutations = nlp, (10.11)

Step 2 Generate random numbers to decide whether mutation is to be performed on a particular population member or not. This is decided by a 'coin toss'. That is, select a random number range to represent true and one to represent false. If the outcome is true, perform mutation, if false, do not.

Step 3 If the outcome found in step 2 is true for a particular population member, then generate another random number to decide the mutation point over the length of the string. Once decided, flip the bit corresponding to the mutation point. With the end of mutation, the strings obtained represent the next generation. The same operations are carried out on this generation until the optimum value is encountered.

8.4 Differential Evolution

8.4.1 Introduction

Differential evolution (DE) is a generic name for a group of algorithms that are based on the principles of GA but have some inherent advantages over GA.

DE algorithms are very robust and efficient in that they are able to find the global optimum of a function with ease and accuracy. DE algorithms are faster than GAS.

GAS evaluate the fitness of a point to search for the optimum. In other words, GAS evaluate a vector's suitability. In DE, this vector's suitability is called its cost or profit

depending on whether the problem is a minimization or a maximization problem.

In GAS only integers are used and the coding is done in binary format; in DE, no coding is involved and floating-point numbers are directly used.

If GAS are to be made to handle negative integers, then the coding becomes complicated since a separate signed bit has to be used; in DE, this problem does not arise at all.

XOR Versus Add

In GAS when mutation is performed, bits are flipped at random with some mutation probability. This is essentially an XOR operation. In DE, direct addition is used.For example, In GA, 16 = 10,000 and 15 = 01111 (after mutation). In DE, 15 As is evident, the computation time for flipping four bits and encoding and decoding the numbers in GAS is far more than the simple addition operationperformed in the case of DE. The question that remains is how to decide how much to add? This is the main philosophy behind DE. = 16 + (-1).

8.4.2 DE at a Glance

As already stated, DE in principle is similar to GA. So, as in GA, we use a population of points in our search for the optimum. The population size is denoted by NP. The dimension of each vector is denoted by D. The main operation is the NP number of competitions that are to be carried out to decide the next generation.

To start with, we have a population of NP vectors within the range of the objective function. We select one of these NP vectors as our target vector. We then randomly select two vectors from the population and find the difference between them (vector subtraction). This difference is multiplied by a factor F (specified at the start) and added to a third randomly selected vector. The result is called the noisy random vector. Subsequently, crossover is performed between the target vector and the noisy random vector to produce the trial vector. Then, a competition between the trial vector and the target vector is performed and the winner is replaced into the population. The same procedure is carried out NP times to decide the next generation of vectors. This sequence is continued till some convergence criterion is met.

This summarizes the basic procedure carried out in differential evolution. The details of this procedure are described below.Assume that the objective function is of D dimensions and that it has to be minimized. The weighting constant F and the crossover constant CR are specified. Refer to Fig. for the schematic diagram of differential evolution.

Step 1 Generate NP random vectors as the initial population: Generate (NP x D) random numbers and linearize the range between 0 and 1 to cover the entire range of the

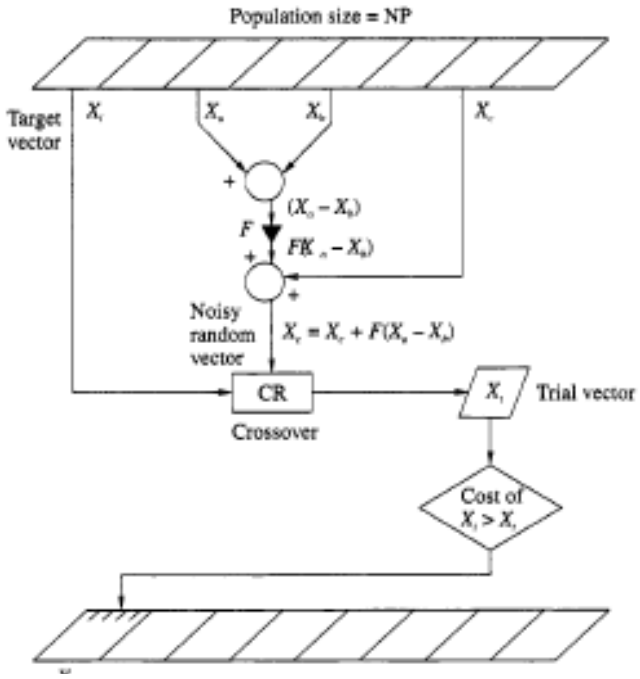


Figure 8.1: Schematic diagram of DE

function. From these (NP x 0)ra ndom numbers, generate NP random vectors, each of dimension D, by mapping the random numbers over the range of the function.

Step 2 Choose a target vector from the population of size NP: First generate a random number between 0 and 1. From the value of the random number decide which population member is to be selected as the target vector (Xi) (a linear mapping rule can be used).

Step 3 Choose two vectors at random from the population and find the weighted difference: Generate two random numbers. Decide which two population members are to be selected (X„ Xb). Find the vector difference between the two vectors (X, - Xb). Multiply this difference by F to obtain the weighted difference.

Weighted difference = F(X, - Xb)

Step 4 Find the noisy random vector: Generate a random number. Choose a third random vector from the population (X,).A dd this vector to the weighted difference to obtain the noisy random vector (Xf).

Noisy random vector (Xf) = X, + F(X, - Xb)

Step 5 Perform crossover between Xi and Xi to find X„ the trial vector: Generate D random numbers. For each of the D dimensions, if the random number is greater than CR, copy the value from Xii nto the trial vector; if the random number is less than CR, copy the value from Xf into the trial vector.

Step 6 Calculate the cost of the trial vector and the target vector: For a minimization problem, calculate the function value directly and this is the cost. For a maximization

problem, transform the objective function Ax) using the rule, F(x) = 1/[1 +Ax)] and calculate the value of the cost. Alternatively, directly calculate the value ofAx) and this yields the profit. In case cost is calculated, the vector that yields the lesser cost replaces the population member in the initial population. In case profit is calculated, the vector with the greater profit replaces the population member in the initial population.

Steps 1-6 are continued till some stopping criterion is met. This may be of two kinds. One may be some convergence criterion that states that the error in the minimum or maximum between two previous generations should be less than some specified value. The other may be an upper bound on the number of generations. The stopping criterion may be a combination of the two. Either way, once the stopping criterion is met, the computations are terminated. Choice of DE Key Parameters (NP, F, and CR) NP should be 5-10 times the value of D, that is, the dimension of the problem. Choose F = 0.5 initially. If this leads to premature convergence, then increase F. The range of values of F is 0 c F c 1.2, but the optimal range is 0.4 c F c 1.0. Values of F c 0.4 and F > 1.0 are seldom effective. CR = 0.9 is a good first guess. Try CR = 0.9 first and then try CR = 0.1. Judging by the speed, choose a value of CR between 0 and 1.

Several strategies have been proposed in DE. A list of them is as follows: (a) DE/ best/l/exp, (b) DE/best/2/exp, (c) DE/best/l/bin, (d) DE/best/2/bin, (e) DE/randtobest/l/exp, (0 DE/rand-to-best/l/bin, (g) DE/rand/l/bin, (h) DE/rand/2/bin,

(i) DE/rand/l/exp, and Q) DE/rand/2/exp.

The notations that have been used in the above strategies have distinct meanings.The notation after the first '/', i.e., best, rand-to-best, and rand, denotes the choice of the vector X,. 'Best' means that X, corresponds to the vector in the population that has the minimum cost or maximum profit. 'Rand-to-best' means that the weighted difference between the best vector and another randomly chosen vector is taken as

X,. 'Rand' means X, corresponds to a randomly chosen vector from the population. The notation after the second '/', i.e., 1 or 2, denotes the number of sets of random vectors that are chosen from the population for the computation of the weighted difference. Thus '2' means the value of X; is computed using the following expression:

where Xal, xbl, Xa2, and xb2 are random vectors. The notation after the third '/', i.e., exp or bin, denotes the methodology of crossover that has been used. 'Exp' denotes that the selection technique used is exponential and 'bin' denotes that the method used is binomial. What has been described in the preceding discussion is the binomial technique where the random number for each dimension is compared with CR to decide from where the value should be copied into the trial vector. But, in the exponential method, the instant a random number becomes greater than CR, the rest of the values are copied from the target vector.

Apart from the above-mentioned strategies, there are some more innovative strategies that are being worked upon by the author's group and implemented on some application.

x: = xc + F(Xal - x b l) + F(Xa2 - xb2,)

Innovations on DE

The following additional strategies are proposed by the author and his associates, in addition to the ten strategies listed before: (k) DE/best/3/exp, (1) DE/best/3/bin, (m) DE/rand/3/exp, and (n) DE/rand/3/bin. Very recently, a new concept of 'nested DE' has been successfully implemented for the optimal design of an auto-thermal ammonia synthesis reactor (Babu et al. 2002). This concept uses DE within DE wherein an outer loop takes care of optimization of key parameters (NP is the population size, CR is the crossover constant, and F is the scaling factor) with the objective of minimizing the number of generations required, while an inner loop takes care of optimizing the problem variables. Yet, the complex objective is the one that takes care of minimizing the number of generations/function evaluations and the standard deviation in the set of solutions at the last generatiodfunction evaluation, and trying to maximize the robustness of the algorithm.

8.4.3 Applications of DE

DE is catching up fast and is being applied to a wide range of complex problems. Some of the applications of DE include the following: digital filter design (Storn 1995), fuzzy-decision-making problems of fuel ethanol production (Wang et al. 1998), design of a fuzzy logic controller (Sastry et al. 1998), batch fermentation process (Chiou & Wang 1999; Wang & Cheng 1999), dynamic optimization of a continuous polymer reactor (Lee et al. 1999), estimation of heat-transfer Parameters in a trickle bed reactor (Babu & Sastry 1999), optimal design of heat exchangers (Babu & Munawar 2000; 2001), Optimization and synthesis of heat integrated distillation systems (Babu & Singh 2000), optimization of an alkylation reaction (Babu & Chaturvedi 2000), optimization of non-linear functions (Babu & Angira 2001a), optimization of thermal cracker operation (Babu & Angira 2001b), scenariointegrated optimization of dynamic systems (Babu & Gautam 2001), optimal design of an auto-thermal ammonia synthesis reactor (Babu et al. 2002), global optimization of mixed integer non-linear programming (MINLP) problems (Babu & Angira 2002a), optimization of non-linear chemical engineering processes (Babu & Angira 2002b; Angira & Babu 2003), etc. A detailed bibliography on DE covering the applications on various engineering and other disciplines is available in literature (Lampinen 2003).

8.5 Interval Mathematics

8.5.1 Introduction

The study of computing with real numbers, i.e. numerical analysis considers exact numbers and exact analysis. Getting an exact representation these exact floating point numbers in a fixed length binary number has been a challenge since inception of computers. With advancement of microprocessor technology and high speed computers, it is now possible to work with higher precision numbers than ever before. In spite of that rounding errors generally occur and approximations are made. There are several other types of mathematical computational errors that can as well affect results obtained using computers. The purpose of interval analysis is to provide upper and lower bounds on the effect all such errors and uncertainties have on a computed quantity.

The scope/ purpose of this section of the workshop are to introduce interval arithmetic concepts and to study algorithms that use interval arithmetic to solve global nonlinear optimization problems. It has been shown that interval global optimization techniques discussed here are guaranteed to produce numerically correct bounds on global optimum value and global optimal point. Even if there are multiple solutions, these algorithms guarantee finding all solutions. It is also guaranteed that solutions obtained are global and not just local.

8.5.2 Interval Analysis

Interval analysis began as a tool for bounding rounding errors (Hansen, 2004). However, it is believed that rounding errors can be detected in another way also. One needs to compute a given result using single precision and double precision, and if the two results agree to some digits then the results obtained are believed to be correct. This argument is however not necessarily valid. Let us look at some examples to understand this in a better way.

Rump's example

Using IEEE-754 computers, the following form (from Loh and Walster (2002)) of Rump's expression with x0 = 77617 and y0 = 33096 replicates his original IBM S/370 results.

f (x, y) =$(333.75 - x2)y^6 + x2(11x2y^2 - 121y^4 - 2) + 5.5y^8 + x/(2y)$

With round-to-nearest (the usual default) IEEE-754 arithmetic, the expression produces:

32-bit: f (x0, y0) = 1.172604

64-bit: f (x0, y0) = 1.1726039400531786

128-bit: f (x0, y0) = 1.172603940053178631858834904520183 8

All three results agree in the first seven decimal digits and thirteen digits agree in the last two results. This may make us think that these results are accurate; nevertheless, they are all completely incorrect. One will be surprised to know that even their sign is wrong. The correct answer is

f (x0, y0) = -0.827396059946821368141165095479816

The reason for such error is probably rounding error and some catastrophic cancellation. The example we have seen here is fabricated and may lead us to think this may not happen in practice. Let us look at some real-life examples where rounding or computational errors caused a disaster.

8.5.3 Real examples

One famous real life example is that of Ariane disaster that happened after just 39 seconds of its launch on 4th June, 1996. Due to an error in the software design (inadequate protection from integer overflow), the rocket veered off its flight path 37 seconds after launch and was destroyed by its automated self-destruct system when high aerodynamic forces caused the core of the vehicle to disintegrate. It is one of the most infamous computer bugs in history.

The European Space Agency's rocket family had an excellent success record, set by the high success rate of the Ariane 4 model. The next generation of Ariane rocket family, Ariane 5, reused the specifications from Ariane 4 software. The Ariane 5's flight path was considerably different and beyond the range for which the reused computer program had been designed. Because of the different flight path, a data conversion from a 64-bit floating point to 16-bit signed integer value caused a hardware exception (more specifically, an arithmetic overflow, as the floating point number had a value too large to be represented by a 16-bit signed integer). Efficiency considerations had led to the disabling of the software handler for this error trap, although other conversions of comparable variables in the code remained protected. This caused a cascade of problems, culminating in destruction of the entire flight.

The subsequent automated analysis of the Ariane code was the first example of large-scale static code analysis by abstract interpretation. It was not until 2007 that European Space Agency could reestablish Ariane 5 launches as reliable and safe as those of the predecessor model.

On February 25, 1991, an Iraqi Scud hit the barracks in Dhahran, Saudi Arabia, killing 28 soldiers from the US Army's 14th Quartermaster Detachment. A government investigation was launched to reveal reason of failed intercept at Dhahran. It turned out that the computation of time in a Patriot missile, which is critical in tracking a Scud,

involved a multiplication by a constant factor of 1/10. The number 1/10 is a number that has no exact binary representation, so every multiplication by 1/10 necessarily causes some rounding error.

$(0.1)10 = (0.00011001100110011...)2$

The Patriot missile battery at Dhahran had been in operation for 100 hours, by which time the system's internal clock had drifted by one third of a second. Due to the closure speed of the interceptor and the target, this resulted in an error of 600 meters. The radar system had successfully detected the Scud and predicted where to look for it next, but because of the time error, looked in the wrong part of the sky and found no missile. With no missile, the initial detection was assumed to be a spurious track and the missile was removed from the system. No interception was attempted, and the missile impacted on a barracks killing 28 soldiers.

Use of standard interval analysis could have detected the roundoff difficulty in the Patriot missile example. The extended interval arithmetic (Hansen, 2004) would have produced a correct interval result in the Ariane space shuttle example, even in the presence of overflow. The Columbia US Space Shuttle maiden flight had to be postponed because of a clock synchronization algorithm failure. This occurred after the algorithm in question had been subjected to a three year review process and formally proved to be correct.

Unfortunately, the proof was flawed. Interval Arithmetic can be used to perform exhaustive testing that is otherwise impractical. All of these and similar errors would have been detected if interval rather than floating-point algorithms had been used.

Performance Benchmark

The relative speed of interval and point algorithms is often the cause of confusion and misunderstanding. People unfamiliar with the virtues of interval algorithms often ask what is the relative speed of interval and point operations and intrinsic function evaluations. Aside from the fact that relatively little time and effort have been spent on interval system software implementation and almost no time and effort implementing interval-specific hardware, there is another reason why a different question is more appropriate to ask. Interval and point algorithms solve different problems. Comparing how long it takes to compute guaranteed bounds on the set of solutions to a given problem, as compared to providing an approximate solution of unknown accuracy, is not a reasonable way to compare the speed of interval and point algorithms. For many even problems, the operation counts in interval algorithms are similar to those in non-interval algorithms. For example, the number of iterations to bound a polynomial root to a given (guaranteed) accuracy using an interval Newton method is about the same as the number of iterations of a real Newton method to obtain the same (not guaranteed) accuracy.

Virtues of Interval Analysis

The admirable quality of interval analysis is that it enables the solution of certain problems that cannot be solved by non-interval methods. The primary example is the global optimization problem, which is the major topic of this talk. Prior to the use of interval methods, it was impossible to solve the nonlinear global optimization problem except in special cases. In fact, various authors have written that in general it is impossible in principle to numerically solve such problems. Their argument is that by sampling values of a function and some of its derivatives at isolated points, it is impossible to determine whether a function dips to a global minimum (say) between the sampled points. Such a dip can occur between adjacent machine-representable floating point values. Interval methods avoid this difficulty by computing information about a function over continua of points even if interval endpoints are constrained to be machine-representable. Therefore, it is not only possible but relatively straightforward to solve the global optimization problem using interval methods.

The obvious comment regarding the apparent slowness of interval methods for some problems (especially if they lack the structure often found in real world problems) is that a price must be paid to have a reliable algorithm with guaranteed error bounds that non-interval methods do not provide. For some problems, the price is somewhat high; for others it is negligible or nonexistent. For still others, interval methods are more efficient.

There are several other virtues of interval methods that make them well worth paying even a real performance price. In general, interval methods are more reliable. As we shall see, some interval iterative methods always converge, while their non-interval counterparts do not. An example is the Newton method for solving for the zeros of a nonlinear equation.

Ease of use and availability of software

Despite several advantages of interval analysis, interval mathematics is less prevalent in practice than one might expect. The main reasons for this are:

i. Lack of convenience (avoidable)
ii. Slowness of many interval arithmetic packages (avoidable)
iii. Slowness of some interval algorithms (occasional)
iv. Difficulty of some interval problems

For programming convenience, an interval data type is needed to represent interval variables and interval constants as single entities rather than as two real interval endpoints. Now-a-days numerous computing tools support computations with intervals. This includes, Mathematica, Maple, MuPad, Matlab, etc. Good interval arithmetic software for various applied problems is now often available (http://cs.utep.edu/interval-

comp/intsoft.html). More recently compilers have been developed by Sun Microsystems Inc. that represents the current state of the art and allow interval data types (e.g. SUN Fortran 95).

8.5.4 Interval numbers and arithmetic

Interval numbers

Interval number (or interval) X is a closed interval, i.e. a set of all real numbers between and including end-points. Mathematically, can be written as:

$$X = \{x | a \leq x \leq b\} \tag{8.1}$$

A degenerate interval is an interval with zero width, i.e. . The endpoints and of a given interval might not be representable on a given computer. Such an interval might be a datum or the result of a computation on the computer. In such a case, we round down to the largest machine-representable number that is less than and round up to the smallest machine-representable number that is greater than . Thus, the retained interval contains . This process is called outward rounding.

An under-bar indicates the lower endpoint of an interval; and an over-bar indicates the upper endpoint. For example, if , then and . Similarly, we write .

Finite interval arithmetic

Let +, -, $\times$, $and \div$ $denote the operations of addition, subtraction, multiplication, and division, respectivel$

$$X \bullet Y = \{x \bullet y | x \in X, y \in Y\} \tag{8.2}$$

Thus the interval X•Y resulting from the operation contains every possible number that can be formed as x • y for each x ■ X, and each y ■ Y. This definition produces the following rules for generating the endpoints

of X • Y from the two intervals X = [a, b] and Y = [c, d].

$$X + Y = [a + c \; b + d] \tag{8.3}$$

$$X - Y = [a - d \; b - c] \tag{8.4}$$

Division by an interval containing 0 is considered in extended interval arithmetic (See Hansen, 2004).

Interval arithmetic operations suffer from dependency and this can result in widening of computed intervals and make it difficult to compute sharp numerical results of complicated expressions. One should always be aware of this difficulty and, when possible, take steps to reduce its effect.

Real functions of intervals

There are a number of useful real-valued functions of intervals, some of them are listed here.

The mid-point of interval is

The width of X is

The magnitude is defined as maximum value of for all

The mignitude is defined as minimum value of for all

The interval version of absolute value of X is defined as

Interval arithmetic is quite involved with varied theorems and criteria to produce sharper bounds on computed quantities. Interested readers are requested to refer Interval Analysis by Moore (Moore, 1966). We now move quickly to interval global optimization techniques.

8.5.5 Global optimization techniques

Unconstrained optimization problem

Here, we consider the problem

where f is a scalar function of a vector x of n components. We seek the minimum value f* of f and the point(s) x* at which this minimum occurs.

For simplicity, we assume that any used derivative exists and is continuous in the region considered. However, Interval methods exist that do not require differentiability. (See Ratschek and Rokne (1988) or Moore, Hansen, and Leclerc (1991))

Interval analysis makes it possible to solve the global optimization problem, to guarantee that the global optimum is found, and to bound its value and location. Secondarily, perhaps, it provides a means for defining and performing rigorous sensitivity analyses.

Until fairly recently, it was thought that no numerical algorithm can guarantee having found the global solution of a general nonlinear optimization problem. Various authors have flatly stated that such a guarantee is impossible. Their argument was as follows: Optimization algorithms can sample the objective function and perhaps some of its derivatives only at a finite number of distinct points. Hence, there is no way of knowing whether the function to be minimized dips to some unexpectedly small value between

sample points. In fact the dip can be between closest possible points in a given floating point number system. This is a very reasonable argument; and it is probably true that no algorithm using standard arithmetic will ever provide the desired guarantee. However, interval methods do not sample at points. They compute bounds for functions over a continuum of points, including ones that are not finitely representable.

Example 4 *Example 1*

Consider the following simple optimization problem

We know that the global minimum will be at +/- 2■■. Suppose we evaluate f at x=1 and over the interval . We obtain,

Thus, we know that f (x) ■ 0 for all x ■ [3, 4.5] including such transcendental points as ■ = 3.14.... Since f (1) = ■3, the minimum value of f is no larger than ■3. Therefore, the minimum value of f cannot occur in the interval [3, 4.5]. We have proved this fact using only two evaluations of f. Rounding and dependence have not prevented us from infallibly drawing this conclusion. By eliminating subintervals that are proved to not contain the global minimum, we eventually isolate the minimum point. In a nutshell, an algorithm for global optimization generally provides various (systematic) ways to do the elimination.

Global optimization algorithm

The algorithm proceeds as follows:

i. Begin with a box x(0) in which the global minimum is sought. Because we restrict our search to this particular box, our problem is really constrained. We will discuss this aspect in later.

ii. Delete subboxes of x(0) that cannot contain a solution point. Use fail-safe procedures so that, despite rounding errors, the point(s) of global minimum are never deleted.

The methods for Step 2 are as follows:

(a) Delete subboxes of the current box wherein the gradient g of f is nonzero. This can be done without differentiating g. This use of monotonicity was introduced by Hansen (1980). Alternatively, consistency methods and/or an interval Newton method can be applied to solve the equation g = 0. In so doing, derivatives (or slopes) of g are used. By narrowing bounds on points where g = 0, application of a Newton method deletes subboxes wherein .

(b) Compute upper bounds on f at various sampled points. The smallest computed upper bound is an upper bound for f*. Then delete subboxes of the current box wherein f > . The concept of generating and using an upper bound in this way was introduced by Hansen (1980).

(c) Delete subboxes of the current box wherein f is not convex. The concept of using convexity in this way was introduced by Hansen (1980).

iii. Iterate Step 2 until a sufficiently small set of points remains. Since x* must be in this set, its location is bounded. Then bound f over this set of points to obtain final bounds on f*.

For problems in which the objective function is not differentiable, Steps 2a and 2c cannot be used because the gradient g is not defined. Step 2b is always applicable. We now describe these procedures and other aspects of the algorithm in more detail.

Initial search region (Initial Box)

The user of the algorithm can specify a box or boxes in which the solution is sought. Any number of finite boxes can be prescribed to define the region of search. The boxes can be disjoint or overlap. However, if they overlap, a minimum at a point that is common to more than one box is separately found as a solution in each box containing it. In this case, computing effort is wasted. For simplicity, assume the search is made in a single box. If user does not specify a box, search is made in default box , where is largest floating point number that can be represented by given number system.

Since we restrict our search to a finite box, we have replaced the unconstrained problem by a constrained one of the form

Actually, we do not solve this constrained problem because we assume the solution occurs at a stationary point of f. However, it is simpler to assume the box X is so large that a solution does not occur at a non-stationary point on the boundary of X.

Walster, Hansen, and Sengupta (1985) compared run time on their interval algorithm as a function of the size of the initial box. They found that increasing box width by an average factor of $9.1 \times 105 increased the run time by an average factor of only 4.4. For one example, however, t$ 108.

Use of gradient

We assume that f is continuously differentiable. Therefore, the gradient g of f is zero at the global minimum. Of course, g is also zero at local minima, at maxima and at saddle points. Our goal is to find the zero(s) of g at which f is a global minimum. As we search for zeros, we attempt to discard any that are not a global minimum of f. Generally, we discard boxes containing non-global minimum points before we spend the effort to bound such minima very accurately.

When we evaluate gradient g over sub-boxes XI, if then there is no stationary point of f in x. therefore x cannot contain the global minimum and can be deleted. We can apply hull consistency to further expedite convergence of the algorithm and sometimes reduce the search box.

An upper bound on the minimum

We use various procedures to find the zeros of the gradient of the objective function. Please note that these zeros can be stationary points that are not the global minimum. Therefore, we want to avoid spending the effort to closely bound such points when they are not the desired solution. In this section we consider procedures that help in this regard.

Suppose we evaluate f at a point x. Because of rounding, the result is generally an interval . Despite rounding errors in the evaluation, we know without question that f I(x) is an upper bound for and hence for f*. Let denote the smallest such upper bound obtained for various points sampled at a given stage of the overall algorithm. This upper bound plays an important part in our algorithm. Since , we can delete any subbox of X for which . This might serve to delete a subbox that bounds a non-optimal stationary point of f. There are various methods that can be applied to the upper bound on the minimum.

Termination

The optimization algorithm splits and subdivides a box into sub-boxes, at the same time sub-boxes that are guaranteed to not contain a global optimum are deleted. Eventually we will have a few small boxes remaining. For a box to be considered as a solution box, two conditions must be satisfied. One,

Where, is specified by user. This condition means that width of a solution box must be less than a specified threshold. Second, we require

Where, is specified by user. This condition guarantees that the globally minimum value f* of the objective function is bounded to within the tolerance .

8.5.6 Constrained optimization

The constrained optimization problem considered here is

Subject to

We assume that function is twice continuously differentiable and constraints and are continuously differentiable.

Now let's understand the John conditions that necessarily hold at a constrained (local or global) minimum. We use these conditions in solving the above constrained optimization problem.

John's condition for the above optimization problem can be written as:

where, are Lagrange multipliers.

The John conditions differ from the more commonly used Kuhn-Tucker-Karush conditions because they include the Lagrange multiplier u0. If we set u0 = 1, then we obtain the Kuhn-Tucker-Karush conditions.

The John conditions do not normally include a normalization condition. Therefore, there are more variables than equations. One is free to remove the ambiguity in whatever way desired. A normalization can be chosen arbitrarily without changing the solution x of the John conditions. If no equality constraints are present in the problem, we can use

Since Lagrange multipliers are non-negative, this assures .

The multipliers vi (i = 1, • • • , r) can be positive or negative. Therefore, a possible normalization is

However, we want our normalization equation that is continuously differentiable so we can apply interval Newton method to solve the John condition. Therefore we use the following normalization equation:

Where, and is the smallest machine representable number.

An Interval Newton method has been proposed to solve the John condition. Solving the John condition is discussed in detail in Hansen and Walster (1990).

For further reading on interval global optimization, readers are encouraged to browse references listed here.

8.5.7 References

Alefeld, G. (1984). On the convergence of some interval arithmetic modifications of Newton's method, SIAM J. Numer. Anal. 21, 363–372.

Alefeld, G. and Herzberger, J. (1983). Introduction to Interval Computations, Academic Press, Orlando, Florida.

Bazaraa, M. S. and Shetty, C. M. (1979). Nonlinear Programming. Theory and Practice,Wiley, NewYork.

Hansen, E. R. (1969). Topics in Interval Analysis, Oxford University Press, London.

Hansen, E. R. (1978a). Interval forms of Newton's method, Computing 20, 153–163.

Hansen, E. R. (1978b). A globally convergent interval method for computing and bounding real roots, BIT 18, 415–424.

Hansen, E. R. (1979). Global optimization using interval analysis-the one dimensional case, J. Optim. Theory Appl. 29, 331–344.

Hansen, E. R. (1980). Global optimization using interval analysis-the multi-dimensional case, Numer. Math. 34, 247–270.

Krawczyk, R. (1983). A remark about the convergence of interval sequences, Computing 31, 255–259.

Krawczyk, R. (1986). A class of interval Newton operators, Computing 37, 179–183.

Loh, E. andWalster, G.W. (2002). Rump's example revisited. Reliable Computing, 8 (2) 245–248.

Moore, R. E. (1965). The automatic analysis and control of error in digital computation based on the use of interval numbers, in Rall (1965), pp. 61–130.

Moore, R. E. (1966).Interval Analysis, Prentice-Hall, Englewood Cliffs, New Jersey.

Moore, R. E. (1979).Methods and Applications of Interval Analysis, SIAM Publ., Philadelphia, Pennsylvania.

Moore, R. E., Hansen, E. R„ and Leclerc, A. (1992). Rigorous methods for parallel global optimization, in Floudas and Pardalos (1992), pp.321–342.

Nataraj, P. S.V. and Sardar, G. (2000). Computation of QFT bounds for robust sensitivity and gain-phase margin specifications, Trans. ASME Journal of Dynamical Systems, Measurement, and Control, Vol. 122, pp. 528–534.

Nataraj, P. S. V. (2002a). Computation of QFT bounds for robust tracking specifications, Automatica, Vol. 38, pp. 327–334.

Nataraj, P. S.V. (2002b). Interval QFT:A mathematical and computational enhancement of QFT, Int. Jl. Robust and Nonlinear Control, Vol. 12, pp. 385–402.

Neumaier, A. (1990). Interval Methods for System of Equations, Cambridge University Press, London.

Ratschek, H., and Rokne, J. (1984). Computer Methods for the Range of Functions, Halstead Press, NewYork.

Walster, G.W. (1988). Philosophy and practicalities of interval arithmetic, pages 309–323 of Moore (1988).

Walster, G.W. (1996). Interval Arithmetic: The New Floating-Point Arithmetic Paradigm, Sun Solaris Technical Conference, June 1996.

Walster, G.W., Hansen, E. R., and Sengupta, S. (1985). Test results for a global optimization algorithm, in Boggs, Byrd, and Schnabel (1985), pp. 272–287.

6.1. Web References

Fortran 95 Interval Arithmetic Programming Reference

http://docs.sun.com/app/docs/doc/816-2462

INTerval LABoratory (INTLAB) for MATLAB

http://www.ti3.tu-harburg.de/~rump/intlab/

Research group in Systems and Control Engineering, IIT Bombay

http://www.sc.iitb.ac.in/~nataraj/publications/publications.htm

Global Optimization using Interval Analysis

http://books.google.com/books?id=tY2wAkb-zLcC&printsec
=frontcover&dq=interval+global+optimization&hl=en&ei=m8QYTLuhO5mMNbuzoccE&sa=
X&oi=book_result&ct=result&resnum=1&ved=0CCwQ6AEwAA#v=onepage&q&f=false

Global Optimization techniques (including non-inverval)

http://www.mat.univie.ac.at/~neum/glopt/techniques.html

List of Interval related softwares

http://cs.utep.edu/interval-comp/intsoft.html

Interval Global optimization

http://www.cs.sandia.gov/opt/survey/interval.html

Interval Methods

http://www.mat.univie.ac.at/~neum/interval.html

List of publications on Interval analysis in early days

http://interval.louisiana.edu/Moores_early_papers/bibliography.html

Chapter 9

Appendix

9.1 Practice session -1

Example 1.1